Universidade Federal Fluminense

MAICON MELO ALVES

An Interference-aware Virtual Machine Placement Strategy for HPC Applications on Clouds

NITERÓI

2016

Universidade Federal Fluminense

MAICON MELO ALVES

An Interference-aware Virtual Machine Placement Strategy for HPC Applications on Clouds

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science. Topic Area: Computer Systems.

Advisor:
LÚCIA MARIA DE ASSUMPÇÃO DRUMMOND

NITERÓI

2016

MAICON MELO ALVES

An Interference-aware Virtual Machine Placement Strategy for HPC Applications on Clouds

Approved by:

Niterói

2016

To my greatest accomplishment: wife, Cristiane, and daughters, Giovana and Camila.

Acknowledgements

Firstly...

Abstract

Cross-interference can negatively affect the performance of High Performance Computing (HPC) applications when executed in clouds. Some papers of the related literature have considered this issue in the proposed Virtual Machine Placement (VMP) strategies, but they have not employed a suitable method for predicting interference. Other VMP strategies consider the minimization of the number of used physical machines. However, none of them consider both problems together. In this paper, we define the Interference-aware Virtual Machine Placement Problem for HPC applications (IVMP) that tackles both problems by minimizing, at the same time, the interference suffered by HPC applications and the number of physical machines used to allocate them. We propose a mathematical formulation for this problem and a strategy based on the Iterated Local Search framework to solve it. Moreover, we also propose a quantitative and multivariate model to predict interference for a varying number of co-located applications. Experiments were conducted in a real scenario by using a real petroleum reservoir simulator and applications from the HPCC benchmark. They showed that our method reduced interference in more 40%, using the same number of physical machines, when compared to the most widely employed heuristics.

Keywords:

List of Figures

3.1 NUMA nodes and processors used in our experiments 22

3.2 Synthetic applications with distinct access profiles 24

3.3 Frequency of cross-applications interference levels 25

3.4 SLLC accumulated access versus interference level 26

3.5 Virtual network accumulated access versus interference level when SLLC was equal to 0.40 27

3.6 Main steps of Multiple Regression Analysis 30

3.7 Frequency of prediction errors 38

3.8 Median prediction errors for each co-location scheme 40

3.9 Prediction errors per co-location scheme 41

3.10 Median prediction errors for each co-location scheme achieved by the extended model 43

3.11 Comparing prediction errors achieved by extended and gold of standard models 45

4.1 VMP 48

4.2 VMP in two-dimension resource space [76] 49

4.3 Example 51

4.4 Recursos disponÃ■veis depois da alocaÃğÃčo de uma mÃąquina virtual . . 51

5.1 ILSvmp vs FF 62

5.2 ILSvmp vs FF 62

5.3 ILSvmp vs FF 63

5.4 ILSvmp vs FF 63

5.5 ILSvmp vs FF 64

5.6 ILSvmp vs FF 64

5.7 Comparing in details results achieved by ILSivmp and heuristcs when alpha=0.7 66

5.8 Comparing sum of interference levels in a real scenario 68

List of Tables

3.1 STREAM kernels 17

3.2 Application template input parameters 19

3.3 Configuration of the machine used in experiments 21

3.4 Generated synthetic applications and the corresponding access profiles when executed in a dedicated machine 21

3.5 Template input parameters chosen to create synthetic applications 23

3.6 Subset of experiments showing the influence of DRAM accumulated access in interference levels 27

3.7 Results of accumulated score and interference levels for a subset of experiments 28

3.8 Description of HPCC applications instances 35

3.9 SLLC, DRAM and virtual network individual scores of real applications executions 36

3.10 Prediction errors for a subset of interference experiments with real applications 38

3.11 Results of accumulated accesses and interference levels for a subset of experiments 40

3.12 My caption 42

3.13 Synhetic applications executed with a lower number of virtual machines . . 42

5.1 Description of notations used in mathematical formulation for IVMP . . . 55

Acronyms and Abreviations

VMP	:	Virtual Machine Placement;
HPC	:	High Performance Computing;
NUMA	:	Non-uniform Memory Access;
SLLC	:	Shared Last Level Cache;
DRAM	:	Dynamic Randmon Access Memory;
WSS	:	Working Set Size;
KVM	:	Kernel-based Virtual Machine;
QPI	:	Quick Path Interconnect;
MUFITS	:	Multiphase Filtration Transport Simulator;
PKTM	:	Pre-stack Kirchoff Time Migration;
HPCC	:	High Performance Computing Challenge;
HPL	:	High Performance Linpack;
DGEMM	:	Double-precision General Matrix Multiply;
PTRANS	:	Parallel Matrix Transpose;
FFT	:	Fast Fourier Transform;
MRA	:	Multiple Regression Analysis;
MPI	:	Message Passing Interface;
ILS	:	Iterated Local Search;
GRASP	:	Greedy Randomized Adaptive Search Procedure;
VNS	:	Variable Neighborhood Search;
VND	:	Variable Neighborhood Descent;
FF	:	First Fit;
BF	:	Best Fit;
WF	:	Worst Fit;
FFD	:	First Fit Decreasing;
BFD	:	Best Fit Decreasing;
WFD	:	Worst Fit Decreasing;
IVMP	:	Interference-aware Virtual Machine Placement Problem for HPC applications;

BP	:	Bin Packing;
VBP	:	Vector Bin Packing;

Glossary

$B_{i,s}$	:	individual access of application i to shared resource s;
$T_{s,m}$	:	accumulated access to a shared resource s in a physical machine m;
A	:	set of applications to be allocated in the cloud environment;
A_m	:	set of applications allocated to physical machine m;
F_m	:	interference level for applications allocated to physical machine m;
L_{i,A_m}	:	slowdown of application i when co-located with set of applications A_m ;
Z	:	sum of interference levels ;
$E_{i,j,s}$	:	similarity factor between applications i and j concerning shared resource s ;
$G_{m,s}$	:	global similarity factor for physical machine m regarding shared resource s ;
R_m	:	concurrency factor for physical machine m ;
M	:	set of available physical machines;
Mem	:	total amount of memory available on each physical machine;
Cpu	:	total amount of CPU available on each physical machine;
m_i	:	amount of memory requested by application i;
c_i	:	amount of CPU requested by application i;
γ_Q	:	interference level experienced by subset of applications Q;
ω	:	maximum interference level achieved in the environment;
$X_{i,j}$	:	equal to 1 whether application i is allocated to machine j and 0, otherwise;
Y_j	:	equal to 1 whether physical machine j is being used and 0, otherwise;
α	:	weight to determine the relevance of each objective of mathematical model;
$M(s)$	:	normalized number of physical machines used in solution s;
$E_{CPU}(s)$	:	percentage of amount of CPU exceeded in solution s;
$E_{MEM}(s)$	:	percentage of amount of main memory exceeded in solution s;
$Z(s)$	:	sum of interference levels for the solution s;
$R^2 - adj$	:	adjusted coefficient of regression;

Contents

1 Introduction **1**

1.1 Objective 4

1.2 Contributions 5

1.3 Thesis Outline 6

2 Related Work **7**

2.1 Cross-application Interference Problem 7

2.2 Virtual Machine Placement Aware of Interference 8

2.3 Virtual Machine Placement Aware of Interference and Usage of Physical Machines 10

3 Cross-application Interference Prediction Model **13**

3.1 Basic Definitions and Assumptions 13

3.2 Generating a Cross-application Interference Dataset 15

3.2.1 Generating Synthetic Applications 16

3.2.2 Synthetic Workload 19

3.2.3 Measuring Cross-applications Interference 24

3.3 A Multivariate and Quantitative Cross-application Interference Prediction Model 29

3.3.1 Multiple Regression Analysis - Main Concepts 29

3.3.2 Building the Interference Prediction Model 31

3.4 Experimental Tests and Results 32

3.4.1 Description of the Used Real Applications 32

3.4.2 Predicting Interference with the Proposed Model 35

3.4.3 Analysis of the Interference Prediction Model 37

3.5 Extending the Interference Prediction Model 39

3.6 Comparing Extended and Gold Standard Prediction Models 44

4 Efficient Usage of Physical Machines 46

4.1 Background 46

4.2 Vector Bin Packing 48

4.3 Proposed Criterion 49

5 An Interference-aware Virtual Machine Placement Strategy 53

5.1 Mathematical Formulation of IVMP 53

5.2 Multistart Iterated Local Search for the IVMP Problem 56

5.3 Experimental Tests and Results 59

5.3.1 Calibrating Parameter Alpha 61

5.3.2 Experiments in a Real Scenario 67

5.4 Conclusion and Future Work 68

6 Conclusion 69

Appendix A – Published Papers 70

Bibliography 71

Chapter 1

Introduction

High Performance Computing (HPC) is broadly used for solving complex problems in scientific fields such as cosmology, molecular dynamics, quantum chemistry, climate and human genetics. HPC enables scientists to extrapolate their knowledge beyond theoretical and experimental fields in order to understand and optimize known scenarios, or even for predicting natural phenomena like storms, earthquakes and tornadoes. Thus, high computational power delivered by HPC systems is essential to enhance human knowledge by helping scientist to answer the most fundamental questions concerning the universe [41] [107].

Apart from scientific computing, HPC has also been employed to solve engineering or business problems like the ones faced by petroleum and automotive industries. Specifically in petroleum industry, HPC is used to improve oil and gas production in a new or developed field or to explore areas where petroleum can be found and recovered [65] [40] [49] [1] [25]. In addition, with the advent of Big Data, HPC becomes indispensable to process the large amount of data continuously gathered by companies, specially those ones belonging to e-commerce segment. From data analysis results, these companies get insights on the customer behavior with the goal to offer products and services that match to its needs and desires [18] [8] [92] [23].

Besides that, HPC has been frequently used to tackle combinatorial optimization problems such as vehicle routing and transportation, resource allocation, scheduling and packing. In this sort of problem, an optimal solution must be identified from a finite set of feasible solutions. Due to the high number of possible solutions, HPC is employed to accelerate search methods like exact procedures, heuristics and meta-heuristics in such a way that a good, maybe optimal, solution can be found in a reasonable time [74] [26] [27] [99].

Typically, HPC applications are executed in dedicated datacenters. In general, these environments are endowed with server machines targeted to performance, besides providing low latency and high bandwidth networks. More specifically, low latency is particularly critical for synchronous and tightly-coupled HPC applications since a single delay in communication layer can affect their entire execution [50] [53] [95] [81]. Moreover, in these environments, HPC applications are singly executed in a given physical machine in order to avoid concurrent access to shared and non sliceable resources. For all these reasons, a specialized and dedicated infrastructure is nowadays the main option to run HPC applications.

However, in the past few years, Cloud Computing has emerged as a promising alternative to execute these applications. This new computational paradigm brings some attractive advantages when compared with a dedicated infrastructure, such as rapid provisioning of resources and significant reduction in operating costs related to energy, software licence and hardware obsolescence [37] [28] [4] [31] [56] [105] [10]. In order to leverage the adoption of clouds for running HPC applications, some initiatives, such as UberCloud, have provided a free and experimental HPC on-demand service, where users can discuss their experiences on using such environment for executing their HPC applications [35] [36].

Despite these advantages, some challenges must be overcome to bridge the gap between performance provided by a dedicated infrastructure and the one supplied by clouds. Overhead introduced by virtualized layer and hardware heterogeneity, for example, affect negatively the performance of HPC applications when executed in clouds. In addition, the absence of a high-performance interconnect can prevent synchronous and tightly-coupled HPC applications to be correctly executed in this environment [4] [42] [78] [19] [61].

Beyond these problems, the performance of HPC applications can be particularly affected by resource sharing policies usually adopted by cloud providers. In general, one physical server can host many virtual machines holding distinct applications. Although virtualization layer provides a reasonable level of resource isolation, some shared resources available in a physical machine, like cache and main memory, cannot be sliced over all applications running in the top of virtual machines. As a consequence, these co-located applications can experience mutual interference, resulting in performance degradation [67] [79] [43] [103].

In face of this cross-application interference problem, some works, such as [43], [55], [113], [13], [54], [20], and [90], proposed Virtual Machine Placement (VMP) strategies to avoid or, at least, reduce the effect that interference imposes in co-located HPC applications. Some of those works just used a static matrix of interference or proposed a naive procedure to determine it. Other works, even proposing more general methods to determine interference, considered just one kind of shared resource, the Shared Last Level Cache (SLLC), or adopted an approach that evaluates only general and subjective characteristics of HPC applications. However, as discussed in Chapter 3, all of these methods are not suitable for determining the interference because this problem is directly related to the amount and similarity of concurrent access to SLLC, DRAM (Dynamic Random Access Memory) and virtual network, and the number of co-located applications.

Moreover, the vast majority of those works proposed VMP strategies that neglected an important issue on clouds: the minimization of the number of used physical machines. Those works did not evaluate this goal because it is, actually, the opposite of reducing interference. Indeed, by minimizing the number of physical machines, a VMP strategy should consolidate all applications in a small number of physical machines, though this approach would increase interference. On the other hand, the interference could be alleviated by spreading out applications in the set of available physical machines, what would rise the number of used physical machines. Nevertheless, minimizing the number of used physical machines is still essential to reduce operational costs related to power consumption and cooling systems, besides allowing cloud provider to promptly satisfy incoming requests for new virtual machines [86] [41] [17].

Considering the importance of minimizing the number of physical machines, two of those works, [43] and [54], proposed VMP strategies that aims to minimize the number of used physical machines when trying to reduce interference. However, those works neither treated both of those problems together nor seek for the effective minimization of interference. In [43], the VMP strategy treats both problems separately by prioritizing one or another objective depending on the class of HPC application. In [54], authors proposed a solution which attempts to minimize the number of physical machines while keeping interference below an user defined threshold. Although conflicting, the minimization of interference and the number of used physical machines can be evaluated together by using a multi-objective approach. Thus, a VMP strategy, aware of interference, can allocate virtual machines in such a way that a better trade-off between minimizing interference and number of used machines can be achieved.

1.1 Objective

In this thesis, we define the Interference-aware Virtual Machine Placement Problem for HPC Applications in Clouds (IVMP), a multi-objective problem that aims to minimize, at the same time, (i) the interference experienced by HPC applications executed in a same physical machine and (ii) the number of physical machines used to allocate them. We introduce a mathematical formulation, that formally defines the problem, and propose a strategy based on the Iterated Local Search (ILS) framework to solve it.

This proposed strategy, called *ILSivmp*, is basically comprised of three phases. Firstly, a greedy constructive method generates, from scratch, an initial solution by trying to reduce, so far as possible, the interference and the number of used physical machines. After creating this initial solution, the *ILSivmp* executes a VND (Variable Neighborhood Descent) procedure to improve the current solution till reach the local optima. Next, the algorithm applies a perturbation in the current local minimum in order to escape from the local optima. Furthermore, at the end of each ILS iteration, the *ILSivmp* evokes the constructive method to create a new solution to be submitted to the ILS. This multistart approach provides the diversification needed to overcome local minimum towards the global optimal solution [71].

In order to predict interference, the *ILSivmp* uses a multivariate and quantitative prediction model also proposed in this work. This interference prediction model takes into account the (i) amount of simultaneous accesses to shared resources, (ii) the similarity between applications access profiles, and (iii) the number of co-located applications to estimate the interference level suffered by a set of co-located applications. More specifically, our proposed model considers the effect that SLLC, DRAM (Dynamic Random Access Memory) and virtual network concurrent accesses incur in cross-application interference. Those three resources are considered particularly critical because (i) SLLC and DRAM are shared among cores of a processor and, (ii) virtual network, although not a hardware resource, is emulated by the hypervisor which is a central component shared by all virtual machines [110] [3]

This cross-interference prediction model was validated by using two real-life HPC applications frequently used in the petroleum industry, namely the Multiphase Filtration Transport Simulator (MUFITS) [2] [24] [83] and the Pre-stack Kirchhoff Time Migration (PKTM) [5], and applications provided by a well-known computing benchmark, the High Performance Computing Challenge (HPCC) [15] [30] [53] [61]. Those applications were executed with different instances as input and results showed that our model predicted cross-application interference with maximum and median errors of 13.88% and 3.59%, respectively. Besides that, the prediction error was less than 10% in 93% of all tested cases.

Concerning the minimization of the number of used physical machines, we adopted in our VMP strategy an approach similar to the one employed by [43]. This approach is based on the Vector Bin Packing problem, a variant of the classical Bin Packing problem, and aims to balance the use of physical resources by considering both the size and shape of the exploitable volume of resources available in the cloud environment [76].

Our proposed VMP strategy was evaluated by using a set of test cases created from the aforementioned validation workload. Results achieved by our strategy were compared with the ones obtained by the most widely used heuristics to minimize the number of machines in clouds. The obtained results indicate that, even using the same number of minimal physical machines given by those heuristics, our strategy was able to reduce the level of interference suffered by co-located applications in more than 40%.

1.2 Contributions

The main contributions of this work are the following:

1. A formal definition and the corresponding mathematical formulation for the Interference-aware Virtual Machine Placement Problem for HPC Applications in Clouds (IVMP). This problem aims to minimize, simultaneously, the interference suffered by applications allocated to the same physical machine and the number of physical machines used to allocate them in the cloud environment.

2. A strategy based on Iterated Local Search (ILS) framework to solve the IVMP problem. Besides being composed of the most common ILS components like perturbation and local search heuristics, the proposed strategy also employs a multistart mechanism. In this mechanism, new solutions, generated from a greedy constructive method, are submitted to the ILS at the end of each iteration. This approach enables our strategy to cover a larger area of the set of feasible solutions.

3. An empirical evaluation presenting the multivariate and quantitative nature of the cross-application interference problem. This experimental analysis was carried out

by using a set of synthetic applications specially created to investigate the cross-application interference problem. This synthetic workload, which was generated from an application template, is composed of applications that put distinct access pressure on shared resources.

4. A multivariate and quantitative model able to predict cross-application interference level by considering (i) the amount of concurrent accesses to SLLC, DRAM and virtual network, (ii) the similarity between the amount of applications accesses, and (iii) the number of co-located applications.

5. An experimental analysis, executed in a real scenario, of both prediction model and VMP strategy by using a real reservoir petroleum simulator and seismic migration algorithm, besides applications from a well-known HPC benchmark.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents related works and discuss the main contributions of our proposal when compared to them. Chapter 3 describe the proposed multivariate and quantitative model for predicting interference, showing its building process and validation. Chapter 4 presents the criterion used by the proposed VMP strategy to minimize the number of used physical machines. In Chapter 5, we define formally the IVMP problem and introduces the proposed VMP strategy to solve it. Finally, conclusions and directions for future work are presented in Chapter 6.

Chapter 2

Related Work

In this chapter, previous papers that proposed interference-aware VMP strategies or just investigated the cross-application interference problem are presented.

2.1 Cross-application Interference Problem

In this section, we describe works that studied the cross-application interference problem with the goal to determine its root cause. Those works, however, did not introduce any VMP strategy based on their proposed methods.

Mury *et al.* [77] argued that cross-application interference could be determined by adopting a classification of applications. Such qualitative approach is based on the *Thirteen Dwarfs* which classifies applications according to computational methods usually adopted in scientific computing. Such classification is not suitable for determining interference because, as described in Section 3.4.3, two applications belonging to Dense Linear Algebra class can present distinct interference levels. Besides that, those results also pointed out that a same application, namely PTRANS (Parallel Matrix Transpose), can present distinct interference levels when solving instances with different sizes. Actually, experiments conducted in [43] also showed this same behavior when applications EP (Embarrassingly Parallel) and LU (Lower-Upper Gauss-Seidel solver)[1] presented distinct interference levels when solving also instances of different sizes. Those findings indicate that a quantitative approach, which considers the amount of accesses to shared resources, could be a better strategy to determine interference than a strategy based on theoretical computation method employed by applications.

[1]Applications EP, LU, CG, IS and FT are provided by the Nas Parallel Benchmark (NPB).

Chi *et al.* [21] evaluated experimentally the degree at which the performance of a guest virtual machine degrades under different combinations of background workloads. These experiments were carried out by using single-threaded applications like basic Linux commands (cat, gzip, gpg, bzip2), memory tests (cachebench) and games (gnugo). These simple applications are quite different from HPC applications because the latter performs a lot of communication through the virtual network. As that paper tested only single-threaded applications, the effect of virtual network access contention in cross-application interference was not properly assessed. However, as described in Section 3.2.3, the level of interference experienced by co-located HPC applications can vary significantly depending on the amount of accumulated access to virtual network. Therefore, the level of dispute over this shared resource should also be considered when predicting interference for this sort of application.

Unlike those works, our proposed model is able to predict interference for HPC applications because it takes into account the amount of concurrent access to virtual network and to other shared resources like SLLC and DRAM. Besides that, we also introduce an interference-aware VMP strategy that uses this model to minimize the overall interference in the cloud environment.

2.2 Virtual Machine Placement Aware of Interference

Works that proposed VMP strategies aiming to alleviate or reduce interference are presented in this section. It is worth to mention that these VMP strategies are not concerned about minimizing the number of used physical machines.

Some works, although presenting interference-aware VMP strategies, did not propose a solution to determine the cross-application interference problem or just introduced a method to alleviate this problem for non-HPC applications.

Yokoyama [113] and Basto [13] accomplished interference experiments by using benchmark applications in order to generate a static interference matrix. Such matrix was further used as the basis for their proposed interference-aware VMP strategies. They evaluated their VMP strategies just using applications previously used in interference experiments. Thus, those papers neither presented a solution to determine interference nor proposed a method to alleviate this problem.

Rameshan *et al.* [90] proposed a mechanism to prevent latency sensitive applications, such as video streaming and web services, from being adversely affected by best-effort batch applications when co-located in a same physical machine. In such proposal, latency sensitive applications report to mechanism whenever they are under interference caused by batch applications like HPC. Then, the mechanism uses information collected in that instant to predict when latency application will suffer interference again. From this prediction, the mechanism throttles the batch application before it imposes interference to latency application one more time. Thus, that work just proposed a way to work around the interference suffered by latency sensitive applications by monitoring the conditions that lead to occurrence of this problem. So, the root of interference problem was not even investigated and, as a consequence, a solution to determine interference suffered by HPC applications was not proposed. Actually, the proposed solution slowdowns HPC applications with the goal to rise the responsiveness of latency sensitive applications.

Other works, on the other hand, investigated the cross-application interference problem in a broader sense and introduced interference-aware VMP strategies built upon their proposed approaches.

Jin *et al.* [55] classified applications in three SLLC access classes called (i) cache-pollution, (ii) cache-sensitive and (iii) cache-friendly. These classes were further used to propose a VMP strategy with goal to alleviate interference by co-locating HPC applications with compatible SLLC access profiles. That work claimed that cache-pollution applications should be preferably co-located with cache-friendly applications rather than being co-located with cache-sensitive ones. However, through some practical experiments they showed that approach may fail. More specifically, EP and IS (Integer Sort), classified as cache-friendly, suffered distinct interference levels when co-located with CG (Conjugate Gradient), categorized as a cache-pollution application. Besides that, although both CG and FT (fast Fourier Transform) were classified as cache-pollution applications, CG did not present interference when co-located with itself, while FT suffered an interference when co-located with CG. These results indicate that a qualitative approach based on SLLC access pattern is not suitable for determining cross-application interference precisely.

Chen *et al.* [20] introduced the cache contention aware VMP problem. This problem aims to minimize performance degradation of virtual machines by reducing SLLC access contention in a physical host. Besides presenting a formal definition of the problem, authors also proposed a heuristic to solve it. Such heuristic tries to minimize the total cache contention degree by avoiding to co-locate virtual machines with intensive cache consumption, i.e., virtual machines that occupy a large space in shared cache. To estimate the total cache contention degree, that work proposed an algorithm based on the concept of stack distance which allows to capture the temporal reuse behavior of an application when executed in a system with a fully or set-associative cache. Although that paper proposed a quantitative solution to determine interference, the proposed method deems SLLC access contention as the unique cause of cross-application interference. However, as discussed in Section 3.2.3, the level of interference suffered by co-located applications is affected by concurrent access to virtual network and DRAM, as well. In addition, their proposed VMP strategy is application-agnostic. This means that the VMP strategy treats each virtual machine as a separate placement problem, that is, it disregards the relation between virtual machines and applications running in the top of them [41]. As a consequence, the VMP strategy may allocate a set of virtual machines, holding the same application, in distinct physical machines. Due to the absence of a high-performance network, this approach could actually worsen the performance of an HPC application by spreading out their virtual machines across the set of physical hosts available in the cloud environment.

In short, those works just introduced the interference problem or proposed methods which are insufficiently accurate to predict interference for HPC applications. In this work, we propose a model able to predict interference for HPC applications with a reasonable prediction error. This model is capable of predicting interference because it takes into account not only the SLLC access contention, but also the amount of concurrent access to other shared and non-sliceable resources like DRAM and virtual network.

Moreover, all of those works neglected the importance of minimizing the number of physical machines used to allocate applications in cloud. Unlike those works, the VMP strategy proposed in this paper aims to minimize, simultaneously, the interference suffered by co-located applications and the number of physical machines needed to allocate them.

2.3 Virtual Machine Placement Aware of Interference and Usage of Physical Machines

In this section, we present papers that proposed interference-aware VMP strategies which also considered the minimization of the number of used physical machines.

Jersak *et al.* [54] modified some of the most used heuristics to minimize the number of physical machines in such a way that those heuristics can consider the cross-interference when performing placement decisions. Thus, authors adapted the heuristics FFD (First Fit Decreasing), BFD (Best Fit Decreasing) and WFD (Worst Fit Decreasing) to observe a maximum interference limit when allocating virtual machines to physical hosts. In order to predict interference, authors devised a simple interference model to be used as a proof of concept with their heuristics. In such model, the level of interference is defined as a function of the number of virtual machines co-executing in a physical machine. So, the model considers that the higher the number of virtual machines, the higher the interference level will be. This naive strategy is not able to determine interference since our experimental results showed that, for the same number of virtual machines, the interference can vary drastically. This happens because interference is actually related to the amount of access to shared resources and not to the number of virtual machines being co-executed in a host. Additionally, as those heuristics just try to not exceed an predefined interference threshold, they did not seek for the effective minimization of the interference present in the cloud environment.

Gupta *et al.* [43] proposed a VMP strategy that attempts to avoid interference or minimize the number of used machines, depending on the class of HPC applications. Tightly coupled synchronous HPC applications, for example, are always allocated to a dedicated physical machine, i.e., a machine that will host only a single application at a time. On the other hand, for the remaining classes of HPC applications, the VMP strategy seeks for the minimization of the number of physical machines, though also observing an acceptable interference criteria. This interference criteria is based on a maximum SLLC accumulated access limit which was defined from experimental analysis. Authors claimed that this SLLC accumulated threshold guarantee an acceptable level of interference for applications co-located in the same physical machine. Although that work proposed a quantitative strategy to alleviate interference problem, only one shared resource, SLLC, was considered. As previously presented in Section 3.2.3, our experiments showed that other shared resources such as virtual network and DRAM can also influence interference and, consequently, should be systematically evaluated. Moreover, their proposed VMP strategy treats, separately, the minimization of interference and the number of used physical machines because, depending on the HPC class, it focus on solving one or another problem. Furthermore, likewise the aforementioned paper, this VMP strategy did not minimize, in fact, the interference since it is just concerned about not trespassing a given interference limit.

Both of those papers proposed VMP strategies that, despite being interference-aware, treated, separately, the problems of minimizing interference and the number of used physical machines. Actually, by looking only at a interference limit, none of them really attempted to minimize the level of interference that arises in a cloud environment. Note that, even within a maximum interference limit, there could be allocation arrangements that would result in lower interference levels. Moreover, those works employed in their VMP strategies a naive or incomplete method to determine interference.

Unlike those works, we propose a VMP strategy which aims to minimize, simultaneously, the level of interference suffered by co-located applications and the number of physical machines by using a quantitative and multivariate prediction model which evaluates concurrent access to three shared and non-sliceable resources. More than reducing interference, our work seeks for effectively minimize this problem by evaluating co-locations which would achieve the lower overall interference level present in the cloud environment.

Chapter 3

Cross-application Interference Prediction Model

Apart from minimizing the number of used physical machines, the proposed VMP strategy aims to either minimize the level of interference suffered by applications that share a common physical machine.

In order to minimize that interference level, the VMP strategy uses an interference prediction model that was built upon a synthetic interference dataset. This dataset is comprised of distinct levels of access to shared resources and the corresponding interference level experienced by the set of co-located applications. From this interference dataset, the model was created by using the Multiple Regression Analysis, a multivariate statistical technique.

Before presenting that interference prediction model, we firstly describe in next section some basic assumptions and definitions used throughout this thesis.

3.1 Basic Definitions and Assumptions

At first, we present an essential assumption made by our proposal in regards of the allocation of HPC applications in the cloud environment.

In this work, we assumed that virtual machines holding one application are not allocated to more than one physical machine, because, in general, high-performance networks are not available in clouds and many HPC applications run slowly and suffer from poor scalability, i.e., they see no performance gain when adding nodes [44] [53] [73] [32] [93] [48]. On the other hand, the potential increasing in the number of processes executed simultaneously in a same physical machine can be verified in the last processors launched by vendors like Intel and AMD[1]. Thus, in the context of this work, the allocation of one application to a physical machine is equivalent to the allocation of all virtual machines, holding that application, to the same physical machine.

We now define formally the amount of individual and accumulated access of applications to shared resources, besides describing metrics used to measure the cross-application interference in the cloud environment.

As described in Equation 3.1, the *individual access* (B) of an application i to a shared resource s is defined as the sum of access of all virtual machines, holding this application, to s, where N_i is the total number of virtual machines hosting i and $V_{i,j,s}$ is the amount of access of virtual machine V_j to a shared resource s (considering the application i). In the context of this work, the amount of access to SLLC and DRAM are measured in terms of millions of references per second (MR/s), while the access to virtual network is expressed as the amount of megabytes transmitted per second (MB/s).

$$B_{i,s} = \sum_{j=1}^{N_i} V_{i,j,s} \tag{3.1}$$

From Equation 3.1, we defined in Equation 3.2 the *accumulated access* (T) as the sum of all individual access (B) to a shared resource s performed by applications co-located in m, where A_m is the set of applications allocated to the physical machine m. This accumulated access represents the total pressure which all co-located applications put in a given shared resource.

$$T_{s,m} = \sum_{\forall i \in A_m} B_{i,s} \tag{3.2}$$

In this work, the negative impact suffered by co-located applications is measured in terms of the interference level. So, the *interference level* (F) experienced by the set of applications allocated to m is calculated as the average slowdown (L) of applications allocated to this physical machine, as described in Equation 3.3.

$$F_m = \frac{\sum\limits_{\forall i \in A_m} L_{i,A_m}}{|A_m|} \tag{3.3}$$

[1]Recently, Intel® launched processor Xeon Platinum 8180M with 28 cores. Thus, a physical server that supports two of these processors, such as Gigabyte® MD61-SC2, provides 56 cores in total. Similarly, the server Supermicro® AS-1023US-TR4 provides 64 cores with two AMD® EPYC 7601.

Particularly, in this thesis, the *slowdown* is expressed as the percentage of additional time spent by one application when it is concurrently executed with other ones. Formally, the slowdown (L) of one application i is equal to the ratio of the execution time achieved by this application when executed concurrently with other ones (K_{i,A_m}) to the time when executed by itself (H_i) minus 1, as shown in Equation 3.4.

$$L_{i,A_m} = \frac{K_{i,A_m}}{H_i} - 1 \tag{3.4}$$

For example, suppose that the execution times of two applications, namely A1 and A2, when executed in a dedicated physical machine, were equal to 60 and 80 seconds, respectively. Suppose also that both these applications, when concurrently executed in that physical machine, spent 100 seconds. In this case, the slowdown of applications A1 and A2 would be, respectively, 67% and 25%. This percentage represents how much additional time these applications needed to complete their executions when co-located in the same physical machine. Thus, the interference level between both of these applications would be equal to 46% which means that they would suffer, in average, 46% of mutual interference when placed in the same physical machine.

From the interference level, we define in Equation 3.5 the *sum of interference levels* (Z) as the sum of interference levels calculated for each physical machine of the cloud environment (M). This metric is used to assess the overall interference of this environment.

$$Z = \sum_{\forall m \in M} F_m \tag{3.5}$$

At last, we assumed that the interference level is equal to zero in cases where just one application is allocated to a physical machine. This assumption is derived from the fact that this isolated application does not contend for any shared resource.

3.2 Generating a Cross-application Interference Dataset

As mentioned before, we built our proposed interference prediction model from an interference dataset that was specially created to investigate the cross-application interference problem.

In order to create that interference dataset, several co-locating experiments were conducted by using applications with distinct access burden. Those applications were generated from a synthetic application template, originally presented in this work.

Next, that proposed application template is described in Section 3.2.1. In Section 3.2.2, we present the synthetic workload generated from application template and in Section 3.2.3 results of concurrent executions of these synthetic applications are presented.

3.2.1 Generating Synthetic Applications

Access contention to shared resources is the main cause of interference suffered by applications co-located in the same physical machine. To study cross-application interference, the vast majority of previous papers employed real applications provided by traditional HPC benchmarks. However, a real application does not allow to control the number of accesses to each shared resource, and consequently, to evaluate systematically the relation between concurrent accesses and the resulting interference.

Thus, we propose an *application template* from which synthetic applications with distinct access levels are created. From this template, we can create an application which puts a high pressure to SLLC, while keeping a low access level to virtual network, for example. Thus, a set of synthetic applications created from that template allows to observe interference in face of different levels of accesses to SLLC, DRAM and virtual network.

In order to represent the usual behavior of HPC applications [97], applications created from the proposed template execute, alternately, two distinct and well-defined phases. The first one, called *Computation Phase*, represents the phase at which the application performs tasks involving calculation and data movement. The other one, namely *Communication Phase*, is the phase where the application exchanges information among computing pairs.

The proposed application template is shown in Algorithm 1. Firstly, the synthetic application executes the *Main Loop* (lines 1 to 13) whose total number of iterations is controlled by parameter *AppIter*. This parameter leverages the total execution time of the application. Next, the synthetic application executes *Computation Phase Loop* (lines 2 to 9) at which it performs the *Computation Phase* with the number of iterations defined by *Comp*. This *Computation Phase* is based on the benchmark STREAM[2] which is widely used to measure the performance of memory subsystems [84] [109]. In order to measure sustainable memory bandwidth, this benchmark executes four simple vector kernels, as

[2]https://www.cs.virginia.edu/stream/

presented in Table 3.1. Because SUM presents the highest tax of memory access, it was chosen to be included in the proposed template.

Algorithm 1 Synthetic application template

```
    Input parameters: AppIter, Comp, AccessStep, WSSCtrl, CompInt, DtAmount,
    Comm
    /* Main Loop*/
 1: for x=1 to AppIter do
        /* Computation Phase Loop*/
 2:     for y=1 to Comp do
            /* Memory Access Loop*/
 3:         for i=1 to WSSCtrl step AccessStep do
 4:             A[i] = B[i] + C[i];
                /*Compute-intensive Loop*/
 5:             for k=1 to CompInt do
 6:                 T = SquareRoot(T);
 7:             end for
 8:         end for
 9:     end for
        /* Communication Phase Loop*/
10:     for z=1 to Comm do
11:         All-to-All-Communication(D,DtAmount);
12:     end for
13: end for
```

The SUM operation is executed by the inner loop *Memory Access Loop* (lines 3 to 8) which is controlled by two input parameters, *WSSCtrl* and *AccessStep*. The first one defines the sizes of vectors A, B and C, and is indirectly used to determine application's Working Set Size (WSS) [57] [43]. A small WSS usually increases application's cache hit ratio because all data needed by it in a given time interval can be entirely loaded in cache. On the other hand, when the application has a WSS greater than cache capacity, its cache hit ratio decreases because all data is fetched from main memory. Thus, WSS can be used to control the cache hit ratio and, consequently, control the number of DRAM references per second also.

Name	**Operation**	**Bytes per Iteration**
COPY	`a[i] = b[i]`	16
SCALE	`a[i] = b[i]*q`	16
SUM	`a[i] = b[i] + c[i]`	24
TRIAD	`a[i] = b[i] + c[i]*q`	24

Table 3.1: STREAM kernels

The second parameter of *Memory Access Loop*, *AccessStep*, controls the step at which the vector elements are accessed. Thus, when *AccessStep* is equal to 1, all elements of vectors A, B and C are accessed consecutively, resulting in a high cache hit ratio. Otherwise, when *AccessStep* is set to a high value, more data is fetched from main memory, resulting in performance degradation. In other words, this parameter provides another way to control the cache hit ratio and to manipulate the number of DRAM references per second.

Thus, the number of DRAM references per second and application's cache hit ratio can be controlled by performing a fine tuning of both parameters *WSSCtrl* and *AccessStep*. When the application presents a high cache hit ratio, DRAM receives few references per second because data is already available in SLLC. Likewise, the number of memory references increases when application presents a low cache hit ratio.

Besides controlling DRAM access, these parameters allows to handle the number of SLLC references per second as well. When the application presents a low cache hit ratio, the number of SLLC references per second decreases. This happens because, before accessing new data, the previous referenced data, not found in SLLC, has to be fetched from DRAM. Consequently, the application's data access rate is reduced, decreasing also the number of SLLC references per second. On the other hand, when the application presents a high cache hit ratio, the number of SLLC references per second increases. Because most data is rapidly fetched from SLLC, application increases the number of memory access requests per second.

After performing the SUM operation (line 4), the synthetic application executes *Compute-intensive Loop* which repeatedly calculates the square root of variable T (line 6). This loop, whose number of iterations is defined by *CompInt*, makes the application more or less compute-intensive. Note that when *CompInt* is set to a high value, the number of memory references decreases. This happens because variable T, being frequently referenced, is kept stored in the first cache level (cache L1), preventing memory subsystem lower levels from being accessed. As a result, SLLC and DRAM references per second decreases. Thus, together with *WSSCtrl* and *AccessStep*, *CompInt* is also used to manipulate the number of DRAM and SLLC references per second.

After *Computing Phase Loop* execution, synthetic application performs *Communication Phase* by executing *Communication Phase Loop* (lines 10 to 12) whose number of iterations is determined by input parameter *Comm*. This phase is based on MPBench benchmark[3], a tool commonly used to evaluate distributed memory systems based on MPI (Message Passing Interface) [34][47]. From all MPI operations tested by this bench-

[3]http://icl.cs.utk.edu/llcbench/mpbench.html

mark, `MPI_Alltoall` was particularly interesting for this work because it is widely used in scientific applications. When using this collective operation, all application's processes send and receive to/from each other the same amount of data [100] [51] [58].

For each *Comunnication Phase Loop* iteration, the application executes *All-to-All-Communication()* function (line 11) that employs `MPI_Alltoall` of a vector D whose size is defined by the input parameter *DtAmount*. So, this input parameter is used to handle the number of bytes transmitted in the virtual network.

Synthetic applications with distinct access profiles can be generated by varying properly all of these aforementioned input parameters, whose descriptions are summarized in Table 3.2.

Input Parameter	**Description**
AppIter	Application's total number of iterations
Comp	Total number of iterations of *Computation Phase*
Comm	Total number of iterations of *Communication Phase*
WSSCtrl	Working set size
AccessStep	Step which vector elements are accessed
CompInt	Total number of iterations of *Compute-intensive Loop*
DtAmount	Amount of data exchanged among processes

Table 3.2: Application template input parameters

Unlike adopting real applications, with this proposed synthetic application template, several applications that put distinct pressure levels to SLLC, DRAM and virtual network can be generated, providing a proper way to investigate systematically the relation between number of accesses and cross-application interference.

3.2.2 Synthetic Workload

In this section, we present the set of synthetic applications generated from the previously presented application template. This synthetic workload is composed of applications with distinct amounts of individual accesses to the three shared resources.

With the goal to comprise a wide range of access burden, we generated applications by considering three target access levels for each of the three shared resources. The amount of individual accesses to each shared resource is expressed by distinct metrics, such as number of references to memory per second or transmitted bytes per second, and the range of those values are also different. To treat those access rates jointly, we normalized those values in an interval between 0.0 and 1.0, where score 1.0 represents the highest possible access rates achieved by an application based on the proposed template, and score 0.0 represents no access. These scores, in our work, represent different access levels to the shared resources. Then, we created applications with *high*, *medium* and *low* access levels to SLLC, DRAM and virtual network, where the high access level corresponds, in our proposal, to the highest access rate to each shared resource, and medium and low access levels correspond to 50% and 10% of this high access rate, respectively.

To generate those applications with these distinct access levels, we varied input parameters of application template and monitored resulted access rates to each shared resource by using the following monitoring tools: PAPI (Performance Application Programming Interface) [22], OProfile [104] and SAR (System Activity Report) [88]. [4]

We executed this set of synthetic applications in a Itautec MX214 server whose configuration details are described in Table 3.3. As illustrated in Figure 3.1, this server is equipped with two NUMA (Non-Uniform Memory Access) nodes interconnected by a QPI (Quick Path Interconnect) of 6.4 GT/s, each NUMA node has 24GB of DRAM memory and is endowed with a Intel Xeon X5675 3.07GHz processor. A processor has six cores that share a 12MB SLLC unit. Moreover, the virtual environment was provided by KVM (Kernel-based Virtual Machine) hypervisor running on top of Ubuntu Server.

Furthermore, we considered that a single virtual machine has one core and 4GB of main memory. By assigning just one core to each virtual machine, we guarantee that all communication among applications processes is performed over the virtual network. Thus, we can stress this shared resource in order to evaluate its influence in the interference suffered by co-located applications.

In our proposal, each application uses six of those virtual machines. We used CPU affinity to deploy half of the virtual machines of the application in each NUMA node, i.e., three virtual machines were deployed in "NUMA Node #1", while the others were deployed in "NUMA Node #2", in a dedicated machine. In this scenario, the access of a synthetic application to shared resources, specifically SLLC and DRAM, is balanced over two NUMA nodes, avoiding self-interference. Moreover, that configuration will be helpful to evaluate cross-application interference as discussed in the next section.

The set of the generated synthetic applications and the corresponding access profiles are presented in Table 3.4, whereas template input parameters, chosen to generate each application, are described in Table 3.5. All applications execute the same number of it-

[4]Remark that, besides Oprofile and PAPI, other monitoring tools such as Perf[102] can also be used to profile applications by collecting the information available in hardware performance counters.

Model	Itautec MX214
CPU	2x Intel Xeon X5675 3.07 GHz
DRAM	48 GB DDR3 1333 MHz
Disk	5.8 TB SATA 3 GB/s
QPI	6,4 GT/s
S.O.	Ubuntu 15.04
Kernel	3.19.0-15
Hypervisor	KVM
Hardware Emulation	Qemu 2.2.0

Table 3.3: Configuration of the machine used in experiments

erations ($AppIter = 25$) and parameters *Comp* and *Comm* were set to ensure that they spent approximately the same amount of time executing *Computation* and *Communication Phases*. Moreover, all scores were rounded to one decimal place. This explains, for example, why applications S1 and S7, although having presented distinct absolute values, were classified in the same SLLC access level.

	Absolute Value			**Score**		
App.	**SLLC**	**DRAM**	**NET**	**SLLC**	**DRAM**	**NET**
S1	1635	4	300	1.00	0.00	0.10
S2	851	61	324	0.50	0.10	0.10
S3	239	41	312	0.10	0.10	0.10
S4	444	444	318	0.30	1.0	0.10
S5	224	224	324	0.10	0.50	0.10
S6	797	240	318	0.50	0.50	0.10
S7	1597	18	2892	1.00	0.00	1.00
S8	890	43	2810	0.50	0.10	1.00
S9	220	49	2910	0.10	0.10	1.00
S10	438	438	2832	0.30	1.00	1.00
S11	214	214	2892	0.10	0.50	1.00
S12	817	241	2838	0.50	0.50	1.00
S13	1575	22	1392	1.00	0.00	0.50
S14	890	52	1362	0.50	0.10	0.50
S15	228	49	1335	0.10	0.10	0.50
S16	438	438	1375	0.30	1.00	0.50
S17	221	221	1404	0.10	0.50	0.50
S18	824	239	1380	0.50	0.50	0.50

Table 3.4: Generated synthetic applications and the corresponding access profiles when executed in a dedicated machine

Applications S1, S7 and S13 achieved the highest SLLC number of references per second. In order to reach this high SLLC individual access, we adjusted the input parameters to ensure that all memory references were directly satisfied by SLLC, which resulted in a

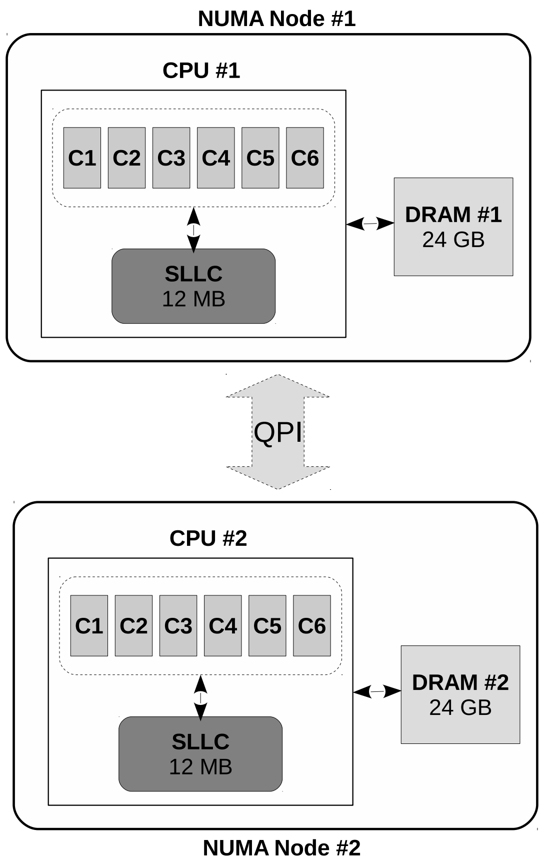

Figure 3.1: NUMA nodes and processors used in our experiments

0.0 DRAM score. Thus, although score 0.0 was not considered as one of the three target access levels, there is no way to achieve the highest number of SLLC references per second without reducing drastically the number of accesses to DRAM.

On the other hand, the number of memory references satisfied by SLLC has to decrease to rise the number of DRAM references per second. To achieve a high DRAM individual access, all memory references should result in accesses to main memory, i.e., the SLLC hit ratio must be close to 0%. However, even in that case, the number of SLLC references per second is not equal to zero, because all references to DRAM are also treated by SLLC. This explains why applications S4, S10 and S16, which achieved the highest number of DRAM references per second, exhibited a SLLC score equal to 0.3.

Thus, concerning the memory subsystem, we were not able to generate all possible combinations involving the three access levels. A high number of individual accesses to SLLC implies in a low number of accesses to DRAM. As a consequence, it is not possible to generate an application which both SLLC and DRAM scores are equal to 1.0 or an application which puts, simultaneously, a high and medium pressure on SLLC and DRAM, for example.

	Parameters						
App.	**AppIter**	**Comp**	**Comm**	**WSSCtrl**	**AccessStep**	**CompInt**	**DtAmount**
S1	25	120000	5200	7000	512	0	22600
S2	25	90000	5200	9000	1024	6	22600
S3	25	40000	5200	11500	2048	22	22600
S4	25	7500	5200	30000	512	0	22600
S5	25	2700	5200	39000	512	21	22600
S6	25	20000	5200	11800	256	2	22600
S7	25	120000	1500	7000	512	0	749568
S8	25	90000	1500	9000	1024	6	749568
S9	25	40000	1500	11500	2048	22	749568
S10	25	7500	1500	30000	512	0	749568
S11	25	2700	1500	39000	512	21	749568
S12	25	20000	1500	11800	256	2	749568
S13	25	120000	150000	7000	512	0	150000
S14	25	90000	150000	9000	1024	6	150000
S15	25	40000	150000	11500	2048	22	150000
S16	25	7500	150000	30000	512	0	150000
S17	25	2700	150000	39000	512	21	150000
S18	25	20000	150000	11800	256	2	150000

Table 3.5: Template input parameters chosen to create synthetic applications

Besides that, that behavior also resulted in some unexpected combinations as the one presented by application S4, that presented DRAM and SLLC scores equal to 1.0 and 0.3, respectively, though this last score was not considered as one of the target access levels.

At last, concerning virtual network, the highest amount of transmitted bytes was achieved by increasing input parameter *DataAmount* up to reaching the maximum amount of data that the hypervisor is able to handle at the same time. We varied *DataAmount* to find out the virtual network saturation threshold. When this limit is exceeded, the amount of bytes transmitted per second decreases, regardless of increasing *DataAmount*.

Although we did not generate all possible combinations of applications, we were able to create a synthetic workload with distinct computational burden. As can be seen in Figure 3.2, this set of synthetic applications comprises a wide range of access profiles, varying from low access applications, such as S3 and S5, to the ones that put, simultaneously, a high pressure on two shared resources such as S7 and S10. We claim that this broad range of access profiles is suitable for conducting a deeper evaluation of the cross-application interference problem.

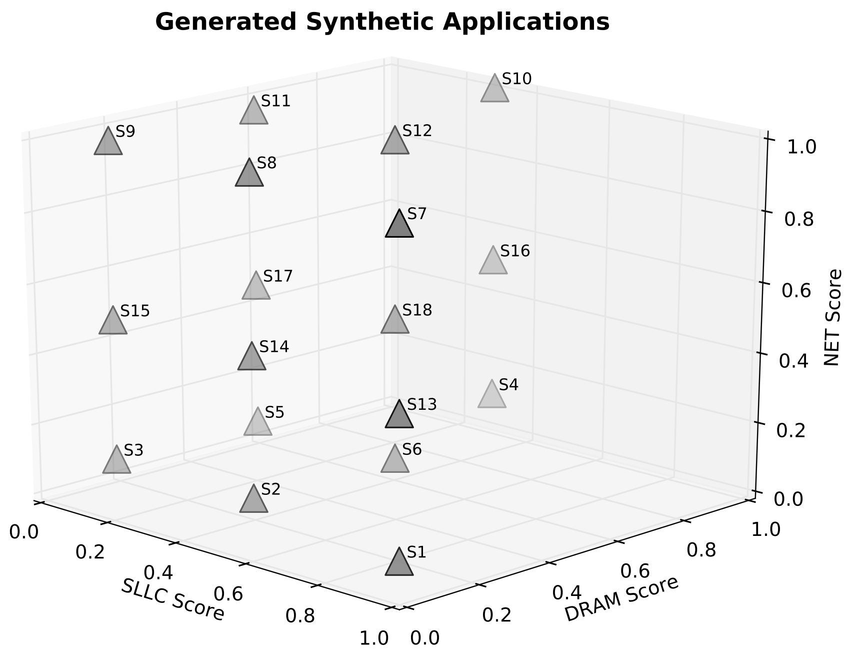

Figure 3.2: Synthetic applications with distinct access profiles

3.2.3 Measuring Cross-applications Interference

In this section, we present experiments to determine cross-applications interference. The previously presented synthetic applications were executed in a two-by-two fashion to obtain the resulting interference level in several cases. Because each synthetic application used half of available resources (memory and CPU), we were able to co-locate two of those applications in the physical machine, without exceeding the available resources in the system. That allocation represents a realistic scenario, usually found in clouds, where all resources available in a physical machine are fully allocated to maximize resource utilization, i.e., to minimize the number of used physical machines in the cloud environment [86] [96].

The generated synthetic applications do not present exactly the same execution time, so to keep concurrency among co-located applications until the end of the experiment, smaller execution time applications were re-started automatically as many times as necessary to cover the entire execution of longer applications. We adopted such approach to fairly measure the interference suffered by both applications, regardless of their execution times.

The overall interference results are summarized in Figure 3.3. Around 54% of the total concurrent executions (93 occurrences) achieved an interference level less than 50%, while

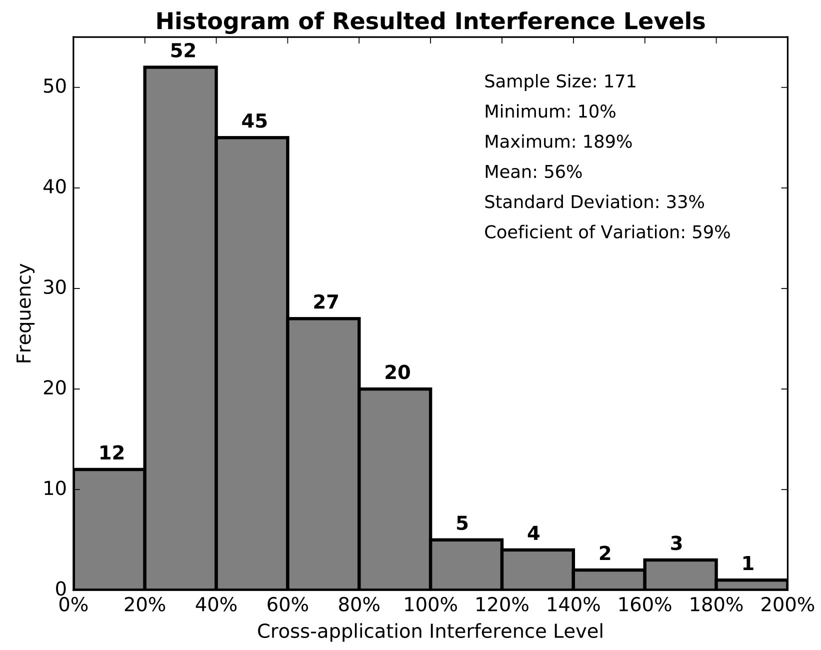

Figure 3.3: Frequency of cross-applications interference levels

37% of co-locations (63 occurrences) suffered interference levels between 50% and 100%. Besides that, in around 9% of all cases (15 occurrences) co-location applications reached interference levels greater than 100%. These results presented a coefficient of variation[5] close to 59% which allows to assert that these synthetic experiments comprised a large range of interference levels.

A deeper analysis accomplished in these results revealed that there is a correlation between interference level and SLLC accumulated access. As can be seen in Figure 3.4, interference level tends to increase as SLLC accumulated access rises. Indeed, the Pearson's correlation coefficient between SLLC accumulated access and interference level is around 0.76, indicating a strong, positive and linear relationship between these both variables. Thus, this observation corroborates the hypothesis that the amount of accesses to shared resources can really influence cross-application interference.

In addition, these experiments allowed to confirm that mutual access to other shared resources besides SLLC can also impact the interference level. Consider, for example, SLLC accumulated score equal to 0.40, in Figure 3.4, it may occur in cases with distinct interference levels. In other words, although SLLC accumulated access presents a strong correlation with interference level, there is, at least, another factor influencing it.

[5]Also known as relative standard deviation, the coefficient of variation is defined as the ratio of the standard deviation to the mean.

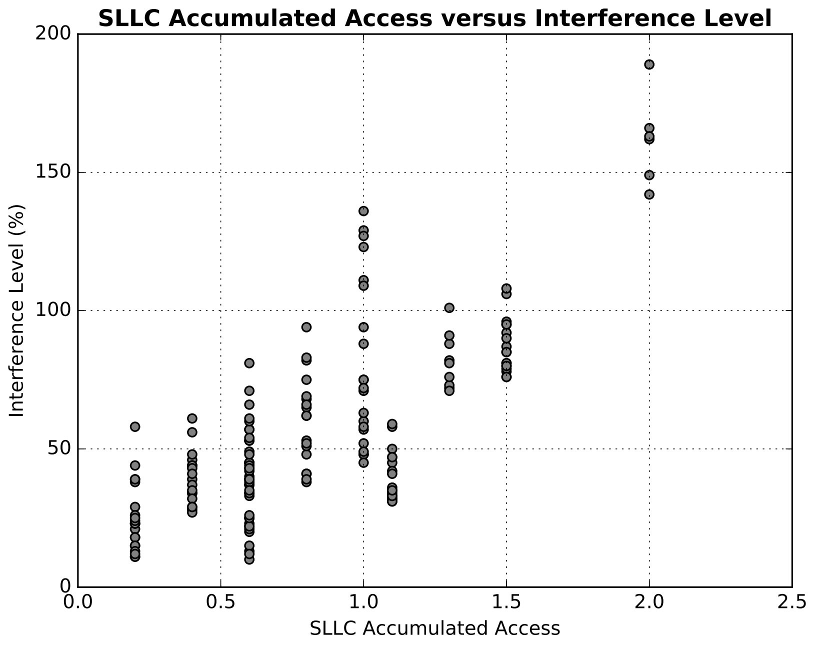

Figure 3.4: SLLC accumulated access versus interference level

Indeed, the interference level is also influenced by the amount of accumulated access to virtual network. As illustrated in Figure 3.5, when virtual network accumulated score is equal to 0.2 and 2.0, the corresponding interference levels are around 28% and 60%, respectively. Thus, for the same SLLC accumulated score, the cross-application interference levels varied more than 30% depending on the amount of access to virtual network.

Similarly, results revealed that concurrent access to DRAM can also impose a negative impact in interference level. As highlighted in Table 3.6, the interference level can change substantially depending on accumulated access to DRAM. For instance, application S2, when co-located with itself, achieved an interference level equal to 71.12%. On the other hand, application S6, even presenting the same SLLC and virtual network access profiles than S2, suffered an interference level around 58% higher than the latter. This same behaviour can be observed for applications S8 and S12. Thus, besides SLLC and virtual network, the accumulated access to DRAM can also affect the level of interference suffered by applications that share a common physical machine.

In addition, some co-locations, even though they presented almost the same amount of accumulated access to the three shared resources, reached interference levels that varied in more than 45%. In the subset of interference results, listed in Table 3.7, for example, the co-location "S14xS8" suffered an interference level 46% higher than the co-location "S15xS7", although both have the same virtual network accumulated score, 1.5, and differ

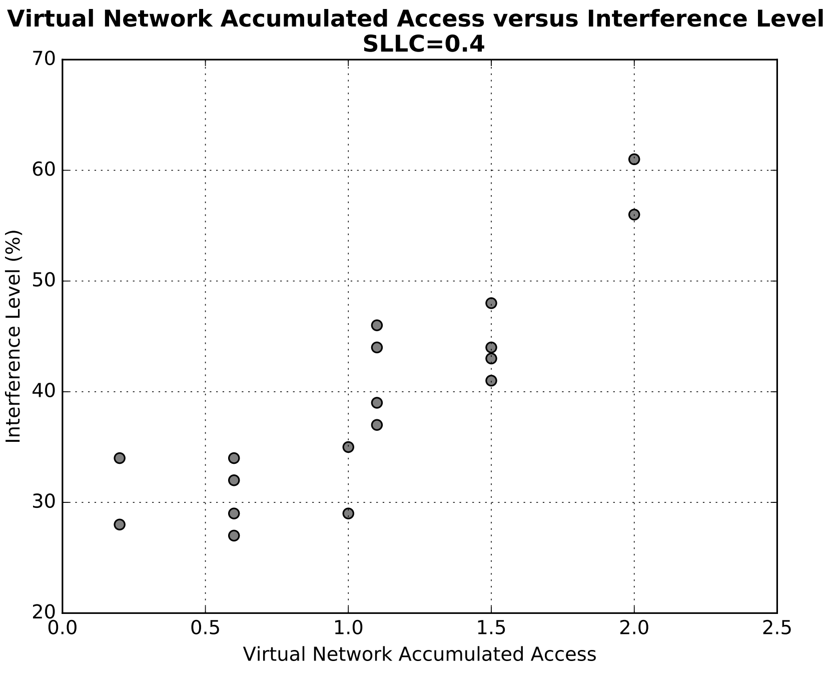

Figure 3.5: Virtual network accumulated access versus interference level when SLLC was equal to 0.40

Co-execution	**Accumulated Access**			**Interference Level**
	SLLC	**DRAM**	**NET**	
S2 x S2	1.0	0.2	0.2	71.12%
S6 x S6	1.0	1.0	0.2	129.28%
S8 x S8	1.0	0.2	2.0	94.02%
S12 x S12	1.0	1.0	2.0	136.09%

Table 3.6: Subset of experiments showing the influence of DRAM accumulated access in interference levels

slightly about DRAM and SLLC accumulated scores. Applications S15 and S7 present distinct SLLC individual access values, the former presents a lower number of SLLC individual access than the latter. This explains why this co-location does not present a high interference level, even achieving a high SLLC accumulated access. On the other hand, the co-location “S14xS8”, although present a SLLC accumulated access similar to co-location “S15xS7”, achieved a high interference level because both applications, which present similar SLLC individual access, evenly compete for SLLC. As can be seen in Table 3.7, this also happens in co-locations that present virtual network accumulated scores equal to 0.6 and 0.2.

So, besides accumulated access to shared resources, the similarity between the amounts of application’s individual access has a direct impact in interference level suffered by applications when co-located in a same physical machine. Notice that this similarity among

Co-execution	**Accumulated Score**			**Interference Level**
	SLLC	**DRAM**	**NET**	
S1 x S3	1.1	0.1	0.2	34.10%
S2 x S2	1.0	0.2	0.2	71.12%
S13 x S3	1.1	0.1	0.6	31.51%
S14 x S2	1.0	0.2	0.6	71.83%
S15 x S7	1.0	0.2	1.5	41.67%
S14 x S8	1.1	0.1	1.5	87.97%

Table 3.7: Results of accumulated score and interference levels for a subset of experiments

applications profiles justifies the difference between interference levels achieved by co-locations listed in Table 3.7.

In order to measure the level of similarity between two applications, we define in Equation 3.6 the *similarity factor* (E). The *similarity factor* of two applications regarding to a shared resource s is calculated as the difference between 1 (highest individual access score) and the absolute value resultant from the difference between the amount of *individual accesses* (B) of applications i and j to a shared resource s.

$$E_{i,j,s} = 1 - |B_{i,s} - B_{j,s}| \tag{3.6}$$

Remark that, because the individual accesses (B) of applications are normalized between 0 and 1, the similarity factor will fall in this same interval. A similarity factor close to 1 indicates a high level of similarity between two applications, while values close to 0 mean that applications present distinct access profiles concerning a shared resource. In other words, the higher the similarity factor is, the higher the level of similarity between two applications will be.

For example, concerning SLLC, the similarity factors for co-locations “S14xS8” and “S15xS7” are equal to 0.1 and 1.0, respectively. These values point out that S15 and S7 present a higher level of similarity than the one presented by S14 and S8. Indeed, as previously discussed, the applications S14 and S8 evenly compete for SLLC, while S15 and S7 put distinct access pressure on this shared resource.

As similarity factor is used to assess similarity between pairs of applications, we devised an additional metric to measure the overall level of similarity for all applications placed in the same physical machine. Then, we define the *global similarity factor* (G) as the average of all similarity factors, concerning a shared resource s, calculated for each pair of applications allocated to the physical machine m, where u is the number of

2-combinations of A_m, as described in Equation 3.7,

$$G_{s,m} = \frac{\sum\limits_{\forall i,j \in A_m / i \neq j} E_{i,j,s}}{u} \tag{3.7}$$

In short, results of the above described experiments showed that the cross-application interference is influenced by the following factors:

- amount of simultaneous accesses to SLLC;
- amount of simultaneous accesses to virtual network;
- amount of simultaneous accesses to DRAM;
- similarity between the amounts of applications' individual accesses to each shared resource;

As presented above, the cross-application interference problem is influenced by four different factors. In face of its multivariate nature, this problem was addressed by using a multivariate statistical technique since it is able to examine the relationship among multiple variables.

3.3 A Multivariate and Quantitative Cross-application Interference Prediction Model

In this section, we describe the proposed prediction model that was created by using the Multiple Regression Analysis technique. In Section 3.3.1, we briefly introduce this statistical method, while in Section 3.3.2 we describe how it was employed to build our prediction model.

3.3.1 Multiple Regression Analysis - Main Concepts

Multiple Regression Analysis (MRA) is a multivariate statistical technique that allows to explain the relationship between one dependent variable and, at least, two independent variables in a dataset. This technique is used to create a model, called statistical variable, able to predict the value of the dependent variable from the known values of independent variables [82] [101].

Basically, MRA comprises four macro steps as illustrated in flowchart of Figure 3.6. Firstly, in *Variables Selection* step, the most likely variables to explain the behavior of the response variable are selected. Although some statistical techniques such as matrix of correlation can provide some insights about which variables must be chosen, this process is mainly guided by the researcher's knowledge about the problem [46] [82].

After that, the *Model Estimation* step is executed, where the terms of the model are determined and their coefficients are automatically estimated by using the Least Squares Method. Next, the *Model Evaluation* step is executed to evaluate goodness-of-fit level, i.e., how satisfactorily the estimated model fits to data used in the building process. Besides that, statistical significance of regression and coefficients are also assessed. In case that estimated model does not present a satisfactory goodness-of-fit level or a desired statistical significance level, a new model should be estimated by executing, again, the *Model Estimation* step. Otherwise, estimated model is ready to be validated in *Model Validation* step by using an "unseen" dataset, i.e, a dataset not used previously during the process of model estimation [46] [82] [72] [106].

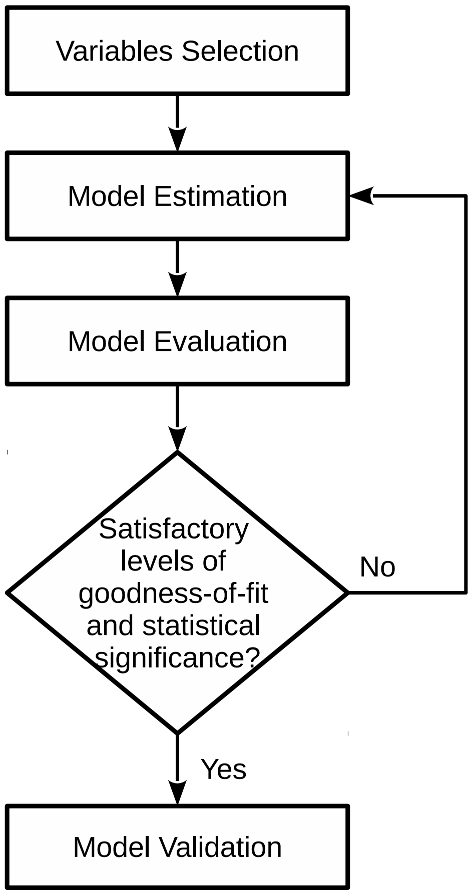

Figure 3.6: Main steps of Multiple Regression Analysis

Although non specialized programs such as Excel and Matlab can be used for building a model from MRA, this process is usually accomplished by using commercial statistical packages such as Eviews [75], SPSS [39], and Minitab [94], or free software ones like Gretl [12] or R [89]. In general, such specific tools provide a full multivariate analysis toolbox that allows to conduct a deep analysis on the estimated model [108] [80].

3.3.2 Building the Interference Prediction Model

By using Minitab version 17.1.0, we followed the aforementioned steps to build our proposed model. At first, we executed the *Variables Selection* step to identify which variables should be selected as independent ones. As synthetic dataset was generated specifically to investigate interference, the selection step was straightforward. We determined accumulated scores and global similarity factors of the three shared resources as independent variables and cross-application interference level as the dependent one. By applying MRA, we expect to generate a model able to predict the interference level from the known values of accumulated scores and global similarity factors.

After *Variable Selection* step, we repeatedly executed both *Model Estimation* and *Model Evaluation* steps until reaching a parsimonious model with satisfactory levels of goodness-of-fit and statistical significance. A parsimonious model is able to perform better out-of-sample predictions because, due to its simplicity, it is not usually overfitted to the sample [106].

Our multivariate and quantitative model for predicting interference is described in Equation 3.8. As can be seen in the prediction model, the global similarity factors, G_{SLLC}, G_{DRAM} and G_{NET}, were employed to weight the influence that accumulated accesses, T_{SLLC}, T_{DRAM} and T_{NET}, impose in interference. Moreover, the total amount of access to DRAM was weighted by SLLC accumulated access since all accesses to DRAM are firstly treated in SLLC.

$$\begin{aligned} F_m = &(0.7498 * T_{SLLC,m} * G_{SLLC,m}) + \\ &(0.1598 * T_{NET,m} * G_{NET,m}) + \\ &(0.1456 * T_{DRAM,m} * G_{DRAM,m} * T_{SLLC,m}) \end{aligned} \tag{3.8}$$

The coefficients calculated for each term, namely 0.7498, 0.1598 and 0.1456, indicate that the interference is more influenced by simultaneous and accumulated access to SLLC, represented in the model by the term $T_{SLLC,m} * G_{SLLC,m}$. However, as previously discussed, the access to virtual network ($T_{NET} * G_{NET}$) and DRAM ($T_{DRAM} * G_{DRAM,m} * T_{SLLC,m}$) can also affect interference because both terms, together, presents a coefficient very close to 0.31. Thus, depending on the value of the SLLC accumulated score, the interference can also be primarily determined by accumulated accesses to DRAM and virtual network.

That model presented an Adjusted Coefficient of Regression (R^2-*adj*) equal to 0.912 which means that 91.2% of variance present in the building sample can be explained through that estimated model. This high R^2-*adj* pinpoints that the generated model is fitted to the building sample, indicating that it is able to give accurate predictions about the depended variable, i.e., interference level.

In addition, we assessed statistical significance of the regression model and coefficients of each term by applying the F hypothesis test. At a level of significance of 0.05, test F revealed that regression and its coefficients can be considered as being statistically significant since resulted *p-values* were very close to 0% (0.004). These results indicate that the probability of each coefficient has been estimated just for this sample is practically 0%. In other words, this model has almost 100% chance to be able to predict interference level for any sample besides that one used in the model building process.

Moreover, an analysis accomplished on residuals showed that the estimated model did not violate any of the MRA basic assumptions. Thus, residuals presented (i) linearity, (ii) homocedasticity and (iii) normal distribution.

3.4 Experimental Tests and Results

In this section, we describe experimental tests accomplished to assess the quality of our proposed prediction model. In Section 3.4.1, we describe the workload used for conducting experimental tests. In Section 3.4.2, we present predictions made by using our model, while in Section 3.4.3 we evaluate how precisely these predictions match to interference levels achieved in real experiments.

3.4.1 Description of the Used Real Applications

In order to evaluate the quality of our prediction model, we carried out experiments by using two real-life applications used in petroleum industry, MUFITS and PKTM, and applications from the High Performance Computing Challenge benchmark (HPCC).

MUFITS is employed by petroleum engineers to study the behavior of petroleum reservoir over time. From simulation results, they can make inferences about future conditions of the reservoir in order to maximize oil and gas production in a new or developed field. Basically, the simulator employs partial differential equations to describe the multiphase fluid flow (oil, water and gas) inside a porous reservoir rock [85] [4]. Reservoir simulation is one of the most expensive computational problems faced by petroleum industry since a single simulation can take days, even weeks, to finish. Computational complexity of this problem arises from the high spatial heterogeneity of multi-scale porous media [65] [115] [45] [83].

PKTM is a seismic migration method that provides a subsurface image from earth. This migrated image, that is generated from a raw seismic section, can reveal geological structures where oil and gas can be detected and further recovered. A raw seismic section must be migrated because the acquisition process can produce a subsurface image that originally does not represent the real geological structure found in the target area. PKTM is one of the most used migration methods in industry due to its simplicity, efficiency, feasibility and I/O flexibility. Similarly to reservoir simulator, the seismic migration process is also computationally expensive and a single execution can take several hours to finish, even when executed in supercomputers [111] [5].

Besides using both of those applications, we have also considered applications available in HPCC, a widely used benchmark for evaluating HPC systems. Although HPCC provides seven kernels in total, we just used the ones that represent real HPC applications or operations commonly employed in scientific computing. A brief description of these four kernels follows [15] [66] [60] [114] [29].

- HPL (High Performance Linpack): solves a dense linear system of equations by applying the Lower-Upper Factorization Method with partial row pivoting. This application, that is usually employed to measure sustained floating point rate of HPC systems, is the basis of evaluation for the Top 500 list [6].

- DGEMM (Double-precision General Matrix Multiply): performs a double precision real matrix-matrix multiplication by using a standard multiply method. Even not being a complex real application, this kernel represents one of the most common operation performed in scientific computing, the matrix-matrix multiplication.

- PTRANS (Parallel Matrix Transpose): performs a parallel matrix transpose. As their pairs of processors communicate with each other simultaneously, this applica-

[6]https://www.top500.org/

tion is a useful test to evaluate the total communications capacity of the network.

- FFT (Fast Fourier Transform): computes a Discrete Fourier Transform (DFT) of a very large one-dimensional complex data vector and is often used to measure floating point rate execution of HPC systems.

We considered, for each application, distinct instances to certify that our proposal is able to predict interference, regardless of the size and characteristics of instance being solved. In MUFITS, we considered instances usually adopted in literature. The first one, labeled here as “I1”, considers a simulation of CO2 injection in the Johansen formation by using a real-scale geological model of the formation. The other one, labeled here as “I2”, is related to the 10th SPE (Society of Petroleum Engineers) Comparative Study. Both instances are available on the MUFITS Web site[7].

Concerning PKTM, we used a synthetic and a real seismic sections, labeled here as “I1” and “I2”, respectively. The synthetic seismic section, and its respective velocity model, were created by using a synthetic seismograph. This sort of seismic section, such as Marmousi and SEG/EAGE models, are commonly used for evaluating geophysical solutions, techniques, and algorithms, because they faithfully represent geological features found in real seismic sections [62] [63] [16] [91] [112]. Even so, we also used a real seismic section composed of data collected from a real seismic data acquisition. This data was acquired in Brazil, but, for confidentiality reasons, we can not inform the exact localization where this acquisition took place.

For HPCC, we considered instances whose details are described in Table 3.8. Remark that, specifically for PTRANS and DGEMM, we created a small instance, called I3, to be executed with only one virtual machine. Instances for HPCC were created by adjusting parameters “#N”, “N” and “NB” which correspond, respectively, to the number of problems, the size of the problem treated by application and the size of the block. Remark that each input parameter has a specific meaning for each application. For example, in case of DGEMM, input parameter “N” is used to set the dimension of matrices to be multiplied, while this same parameter, in case of FFT, determines the size of vector of real numbers to be transformed to the frequency domain. More detailed information about these input parameters can be found on the HPCC Web site[8].

[7]http://www.mufits.imec.msu.ru/

[8]http://icl.cs.utk.edu/hpcc/

Application	**Instance**	**HPCC Parameters**		
		#N	**N**	**NB**
HPL	I1	1	18000	80
	I2	1	15000	80
DGEMM	I1	1	3000	80
	I2	1	18000	80
PTRANS	I1	1	24000	80
	I2	5	500	80
FFT	I1	1	65000	10
	I2	1	40000	10

Table 3.8: Description of HPCC applications instances

3.4.2 Predicting Interference with the Proposed Model

With the goal of assessing the prediction model in distinct scenarios, we considered four co-location schemes: A, B, C and D. In schemes A and B, all applications, solving both I1 and I2, were co-located in a two-by-two and three-by-three fashions, respectively. In scheme C, we co-located all applications, except PKTM, in a six-by-six fashion, but solving just I1. At last, in D we co-located applications PTRANS and DGEMM in a twelve-by-twelve fashion when solving I3. In addition, each virtual machine used in all co-location schemes has the same configuration as described in Section 3.2.2 and, as performed in synthetic experiments, half of virtual machines allocated to each application was deployed to a NUMA node.

As presented before, the prediction model relies on the amount of individual access to shared resources to estimate the cross-application interference. So, to test our proposed model, we firstly executed each of those applications in a dedicated physical machine in order to acquire their corresponding access profiles to the three shared resources. This profiling was executed in the server previously described in Section 3.2.2.

Individual access scores achieved by each application are described in Table 3.9. FFT achieved the highest access rate to virtual network, while PTRANS, when solving I1, imposed the highest pressure to SLLC and DRAM. Moreover, as DGEMM and PKTM are embarrassingly parallel applications, they presented a low access level to virtual network. It is worth mentioning that some applications decreased their access rates to shared resources when executing with a lower number of virtual machines. This is expected since the same application, when executed with a lower number of processes, decreases, usually, the amount of individual access to shared resources.

Considering the individual profile of each application, we applied our model to pre-

App. Label	**Application**	**Instance**	**Virtual Machines**	**Individual Access Score**		
				SLLC	**DRAM**	**NET**
MUFITS.I1.P6	MUFITS	I1	6	0.053	0.127	0.004
MUFITS.I2.P6	MUFITS	I2	6	0.030	0.000	0.008
MUFITS.I1.P4	MUFITS	I1	4	0.049	0.075	0.003
MUFITS.I2.P4	MUFITS	I2	4	0.024	0.000	0.006
MUFITS.I1.P2	MUFITS	I1	2	0.016	0.027	0.000
HPL.I1.P6	HPL	I1	6	0.026	0.062	0.015
HPL.I2.P6	HPL	I2	6	0.028	0.057	0.019
HPL.I1.P4	HPL	I1	4	0.018	0.041	0.014
HPL.I2.P4	HPL	I2	4	0.012	0.038	0.011
HPL.I1.P2	HPL	I1	2	0.008	0.011	0.005
DGEMM.I1.P6	DGEMM	I1	6	0.017	0.021	0.000
DGEMM.I2.P6	DGEMM	I2	6	0.010	0.024	0.000
DGEMM.I1.P4	DGEMM	I1	4	0.011	0.023	0.000
DGEMM.I2.P4	DGEMM	I2	4	0.007	0.016	0.000
DGEMM.I1.P2	DGEMM	I1	2	0.004	0.009	0.000
DGEMM.I3.P1	DGEMM	I3	1	0.003	0.005	0.000
PTRANS.I1.P6	PTRANS	I1	6	0.183	0.214	0.322
PTRANS.I2.P6	PTRANS	I2	6	0.018	0.000	0.111
PTRANS.I1.P4	PTRANS	I1	4	0.136	0.091	0.194
PTRANS.I2.P4	PTRANS	I2	4	0.015	0.000	0.038
PTRANS.I1.P2	PTRANS	I1	2	0.054	0.041	0.189
PTRANS.I3.P4	PTRANS	I3	1	0.050	0.029	0.038
FFT.I1.P4	FFT	I1	4	0.068	0.169	0.493
FFT.I2.P4	FFT	I2	4	0.070	0.164	0.517
FFT.I1.P2	FFT	I2	2	0.041	0.086	0.306
PKTM.I1.P6	PKTM	I1	6	0.004	0.000	0.001
PKTM.I2.P6	PKTM	I2	6	0.001	0.000	0.001
PKTM.I1.P4	PKTM	I1	4	0.003	0.000	0.001
PKTM.I2.P4	PKTM	I2	4	0.001	0.000	0.001

Table 3.9: SLLC, DRAM and virtual network individual scores of real applications executions

dict what would be the interference suffered by those applications if they were co-located according to the aforementioned schemes. Prediction results point out that the minimum and maximum predicted interference levels would be equal to 0.20% and 49.60%, respectively. The minimum interference level was predicted for co-location "PKTM.I2.P6xPKTM.I2.P6", while the maximum one was estimated for FFT.I1.P2 when co-located with itself in a six-by-six fashion. As expected, the lowest and highest interference levels were predicted for co-locations that involved, respectively, applications with low and high access rates to SLLC, DRAM and virtual network. In other words, our model indicated that PKTM, DGEMM and HPL would suffer a low cross-interference, while FFT and PTRANS would present the highest interference levels.

However, specifically to PTRANS, our model points out that interference suffered by this application would vary significantly depending on the instance being solved. Our model indicated that PTRANS.I1.P6 would experience an interference level around 40% when co-located with itself in a two-by-two fashion, while PTRANS.I2.P6, when co-located in same conditions, would present an interference level of approximately 12%.

In addition, our model predicted that PTRANS and DGEMM, although belonging to the same Dwarf class, namely Dense Linear Algebra, would present distinct interference levels when co-located with themselves. So, prediction results indicated that interference level resulted from co-location “PTRANS.I1.P6xPTRANS.I1.P6” would be approximately equal to 40%, while “DGEMM.I1.P6xDGEMM.I1.P6” would suffer an interference level close to 3%.

Furthermore, our model predicted that FFT.I1.P4 and MUFITS.I1.P4, although presenting similar SLLC access rates, would suffer distinct interference levels when co-located with themselves in a three-by-three fashion. Specifically, our solution predicted that FFT.I1.P4 would present a mutual interference around 40%, while MUFITS.I1.P4 would suffer a cross-interference approximately equal to 12%.

At last, although all co-locations presented exactly the same total number of running virtual machines (twelve), the model predicted that those applications would experience distinct levels of interference.

3.4.3 Analysis of the Interference Prediction Model

In order to evaluate the quality of our proposed model, we executed all of the co-locations defined in schemes A, B, C and D, and, for each co-location, we calculated the *prediction error* achieved by our model. The prediction error is defined as the absolute value of the difference between interference level predicted by our model and the real interference level suffered by applications.

As can be seen in Figure 3.7, our model presented a median prediction error equal to 3.91%, and, in approximately 92% of all tested cases, error was less than 10%. Remark that, in only 0.75% of all co-locations (5 occurrences), the prediction error was higher than 15%. Such results showed that our model was able to predict, for several co-locations, the interference level with a reasonable prediction error. Thus, the model was capable of estimating interference level for a set of real HPC applications with heterogeneous access profiles and computation patterns.

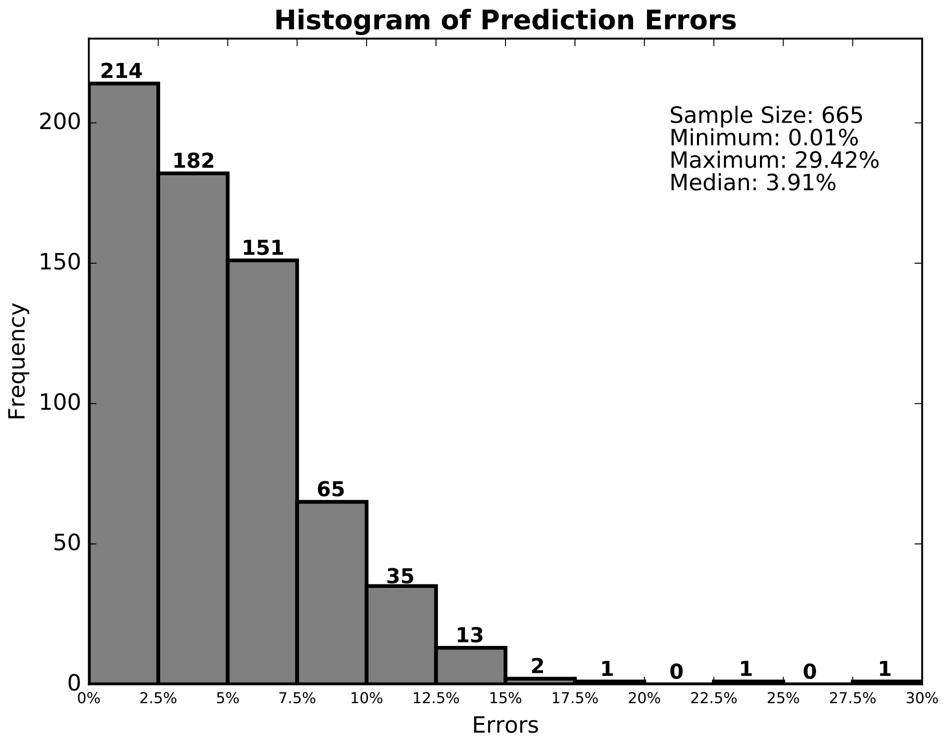

Figure 3.7: Frequency of prediction errors

Some interesting results are highlighted in Table 3.10. At first, as predicted by our model, experimental results showed that PTRANS, when executing with six virtual machines, really presented distinct interference levels when treating I1 and I2. Our model was able to predict interference of PTRANS regardless of the instance being solved because it takes into account the amount of accesses of applications to shared resources. Indeed, as can be seen in Table 3.9, the amount of individual access of PTRANS to all shared resources, when solving I1, is significantly higher than the one achieved when treating I2.

Co-location	**Interference Level**		**Prediction**
	Real	**Predicted**	**Error**
PTRANS.I1.P6xPTRANS.I1.P6	44.50%	39.97%	4.53%
PTRANS.I2.P6xPTRANS.I2.P6	5.31%	12.11%	6.80%
PKTM.I2.P6xPKTM.I2.P6	0.03%	0.20%	0.17%
DGEMM.I1.P6xDGEMM.I1.P6	7.79%	2.50%	5.29%
FFT.I1.P4xFFT.I1.P4xFFT.I1.P4	49.31%	40.45%	8.87%
MUFITS.I1.P4xMUFITS.I1.P4xMUFITS.I1.P4	22.85%	11.65%	11.20%

Table 3.10: Prediction errors for a subset of interference experiments with real applications

Besides that, those experimental results confirmed that PTRANS and DGEMM, even belonging to the same Dwarf class, presented distinct interference levels. This result allows to state that a qualitative approach based on Dwarfs classes is not enough to precisely determine interference. In the other hand, our model was able to predict that PTRANS and DGEMM would present, respectively, a high and a low interference levels.

Moreover, results also showed that SLLC access contention is not the only cause for the cross-interference problem. FFT.I1.P4 and MUFITS.I1.P4, although having a similar amount of SLLC individual accesses, suffered distinct interference levels. As depicted in Table 3.10, our model predicted satisfactorily interference suffered by these applications because it evaluates not only SLLC, but accesses to DRAM and virtual network as well. Indeed, although both applications present similar SLLC access rates, FFT.I1.P4, that suffered a higher interference, puts a higher pressure on virtual network than MUFITS.I1.P4.

Furthermore, results confirmed that, despite the same total number of running virtual machines, our model correctly predicted that applications would experience distinct interference levels. Thereby, the number of virtual machines running in a physical machine can not be used as a parameter to determine the level of interference suffered by applications allocated to the same physical machine.

All these findings allow to assert that solutions based just on (i) number of running virtual machines, (ii) Dwarfs classes or (iii) SLLC access contention are not suitable for determining interference. Our proposed model satisfactorily predicted interference for these specific cases because it takes into account the amount of simultaneous access to SLLC, DRAM and virtual network, besides considering similarities between applications access profiles.

3.5 Extending the Interference Prediction Model

Although that prediction model has achieved satisfactory results, we observed that error increases as the number of co-located applications also does. As shown in Figure 3.8, the median prediction error calculated just for co-locations of scheme A was equal to 3.10%, whereas in scheme D the error raised to 7.95%. Besides that, as depicted in Figure 3.9, prediction errors higher than 15% occurred exclusively in schemes C and D, that is, for cases which six or twelve applications were co-located in the same physical machine. Our model presented high prediction errors for these cases because it was built upon a dataset comprised only of results obtained from two-by-two fashion experiments. In other words, we supposed that only the accumulated access to shared resources and similarities among applications profiles would be enough to predict interference, regardless of the number of co-located applications.

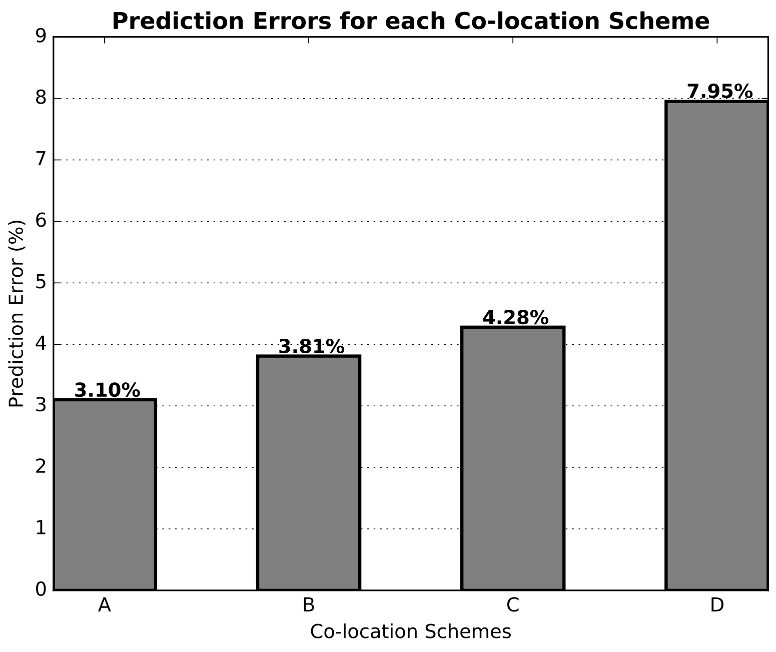

Figure 3.8: Median prediction errors for each co-location scheme

However, as can be seen in Table 3.11, the interference level is either affected by the number of applications co-located in the same physical machine. Consider, for example, applications MUFITS.I1.P6 and MUFITS.I1.P2 when co-located with themselves in a two-by-two and six-by-six fashion, respectively. Although co-location “6 x MUFITS.I1.P2” has a smaller amount of accumulated access to shared resources than “2 x MUFITS.I1.P6”, the former presented a higher interference level than the latter. This happens because the level of concurrency in co-location “6 x MUFITS.I1.P2” is higher than in “2 x MUFITS.I1.P6”, that is, in the first co-location there are four more applications disputing over shared resources than in the second one.

Co-location	Number App.	Accumulated Access			Interference Level
		SLLC	**DRAM**	**NET**	
2 x MUFITS.I1.P6	2	0.11	0.25	0.02	10.72%
6 x MUFITS.I1.P2	6	0.10	0.16	0.00	22.95%

Table 3.11: Results of accumulated accesses and interference levels for a subset of experiments

Therefore, apart from the amount of accumulated access to shared resources and similarity among applications profiles, this level of dispute over shared resources must

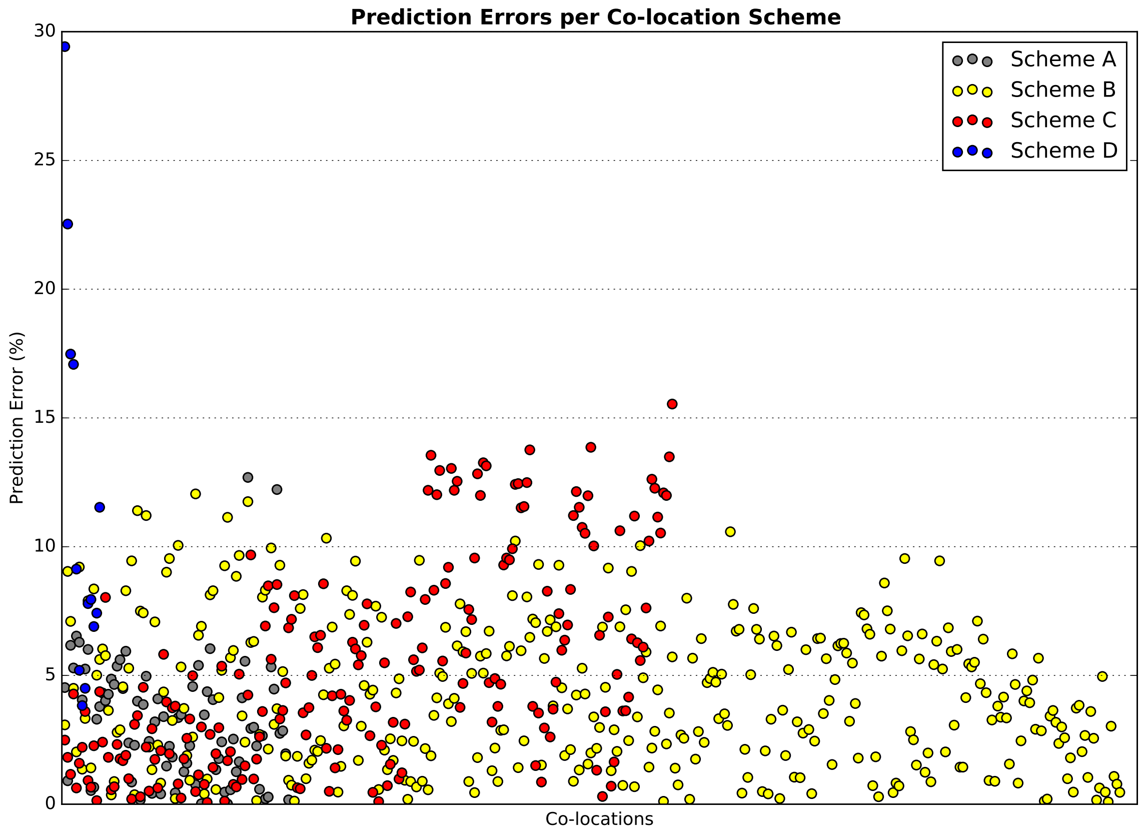

Figure 3.9: Prediction errors per co-location scheme

also be considered when predicting interference for a set of co-located applications. In order to measure this level of concurrency, we defined in Equation 3.9 the *concurrency factor* (R) in a physical machine m as the ratio between the number of applications allocated to m ($|A_m|$) minus 1 and the maximum number of applications that can be allocated to the physical machine (n) minus 1.

$$R_m = \frac{|A_m| - 1}{n - 1} \tag{3.9}$$

As this factor is used to measure the level of dispute over shared resources, it is calculated only for cases which more than one application are allocated to the same physical machine. Anyway, when just one single application in running in the physical machine, the prediction model is neither used since interference level, for those cases, is assumed to be equal to zero, as already stated in Section 3.1.

Aiming to incorporate this level of concurrency in the prediction model, we generated an extended interference dataset comprised of distinct values for the concurrency factor. This dataset was created by varying the number of synthetic applications co-located in the same physical machine. In order to vary this number of co-located applications, we

Table 3.12: My caption

		Individual Access Score		
Application	**Number of Virtual Machines**	**SLLC**	**DRAM**	**NET**
S1	4	1.008	0.000	0.113
S3	4	0.119	0.009	0.109
S4	4	0.257	0.867	0.090
S7	4	0.922	0.036	0.495
S10	4	0.262	0.892	0.487
S12	4	0.713	0.059	0.494
S13	4	0.954	0.000	0.337
S16	4	0.257	0.867	0.326
S18	4	0.717	0.072	0.352
S3	2	0.066	0.000	0.026
S7	2	0.571	0.000	0.279
S10	2	0.502	0.070	0.309
S18	2	0.482	0.000	0.177
S3	1	0.033	0.000	0.000
S7	1	0.282	0.000	0.000

Table 3.13: Synhetic applications executed with a lower number of virtual machines

configured some applications from synthetic workload to be executed with a lower number of virtual machines, as listed in Table 3.13. By using applications with 6, 4, 2 and 1 virtual machines, we could co-locate 2, 3, 6, and 12 applications in the same physical machine to accomplish several co-locating experiments along the lines of schemes A, B, C and D. So, we created the extended interference dataset by calculating, for each co-location, (i) the resulting interference level, (ii) the concurrency factor, (iii) global similarity factors and (iv) accumulated amount of access to shared resources.

From this new dataset and considering concurrency factor as an additional independent variable, we applied the MRA building process to create the extended interference prediction model whose terms are described in Equation 3.10. This parsimonious and statistically significant model presented a R^2-*adj* around 94.55% which is higher than the one achieved by the previous model (91.21%). In regards of model terms, the main difference from the previous model to this one is the use of concurrency factor to weight the influence of SLLC accumulated access in interference level $(T_{SLLC,m} * R_m)$.Thus, depending on the number of co-located applications, the interference level can be even more affected by the amount of SLLC accumulated access.

$$\begin{aligned} F_m = & (0.5680 * T_{SLLC,m} * R_m) + \\ & (0.6758 * T_{SLLC,m} * G_{SLLC,m}) + \\ & (0.1422 * T_{NET,m} * G_{NET,m}) + \\ & (0.0516 * T_{DRAM,m}^2 * G_{DRAM,m}) \end{aligned} \tag{3.10}$$

By using this extended model, we repeated validation experiments conducted with real applications, i.e., we used the extended model for predicting interference levels according to co-location schemes A, B, C and D, and, for each co-location, we calculated the corresponding prediction error. Extended model achieved a median prediction error equal to 3.59% and, in approximately 93% of all tested cases, error was less than 10%.

Moreover, the maximum error achieved by the extended model was equal to 13.88% which is significantly lower than the one presented by the previous model, around 29.42%. Besides that, as can be seen in Figure 3.10, the median prediction error remains almost the same for all co-location schemes. In short, those results indicate that, unlike previous model, the extended model was able to deal properly with the negative effect that the number of co-located applications induces in interference level.

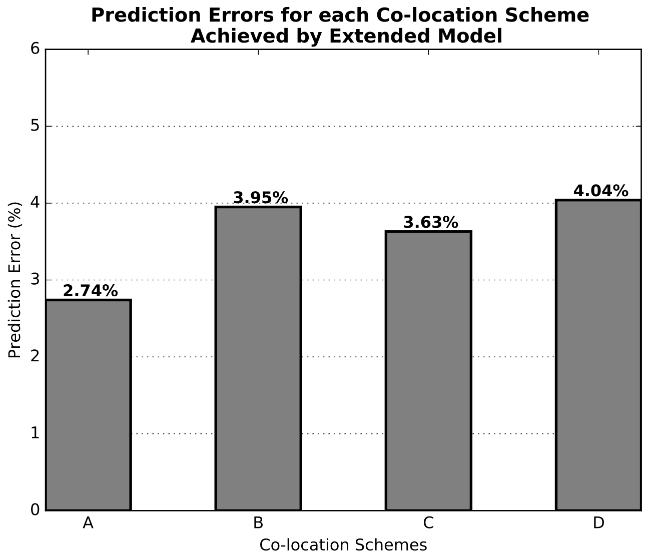

Figure 3.10: Median prediction errors for each co-location scheme achieved by the extended model

3.6 Comparing Extended and Gold Standard Prediction Models

Results described in previous section showed that the extended model was able to predict the interference level suffered by a set of real HPC applications. Despite those satisfactory results, we devised an additional test to evaluate the precision of that extended model by comparing it with a gold standard model. This model was generated exclusively from the dataset provided by real experiments. Because it was built from the validation dataset, we suppose that this gold standard model provides the best predictions for this sample. From this test, we aim to evaluate how close the extended model are from the gold standard model in terms of prediction error.

Thus, we applied the MRA building process in the dataset provided by the real experiments in order to create the gold standard model described in Equation 3.11. This model presented the same terms of the extended model and achieved an R^2-*adj* around 94.13%, practically the same of the extended model (94.55%).

$$\begin{aligned} F_m = &(0.5680 * T_{SLLC,m} * R_m) + \\ &(0.6758 * T_{SLLC,m} * G_{SLLC,m}) + \\ &(0.1422 * T_{NET,m} * G_{NET,m}) + \\ &(0.0516 * T^2_{DRAM,m} * G_{DRAM,m}) \end{aligned} \tag{3.11}$$

As can be seen in Figure 3.11, the gold standard model achieved a median prediction error equal to 3.18% which is smaller than the one presented by the extended model. This is expected since the gold standard model was built and validated by using the same dataset, while the extended model was validated from an unseen dataset. Moreover, the interquartile range[9], calculated for both models, revealed that results obtained from gold standard model presented a smaller data dispersion than extended model. This means that values are concentrate around the median prediction error which confirms the reliability of this value to represent central tendency of data.

Although the gold standard model presented a lower median prediction error than the extended model, the difference between median prediction errors for both models is not significant, i.e., less than 0.5%. Likewise, the small difference between the interquartile

[9]The interquartile range is a measure of statistical dispersion calculated from the difference between the first and third quartiles.

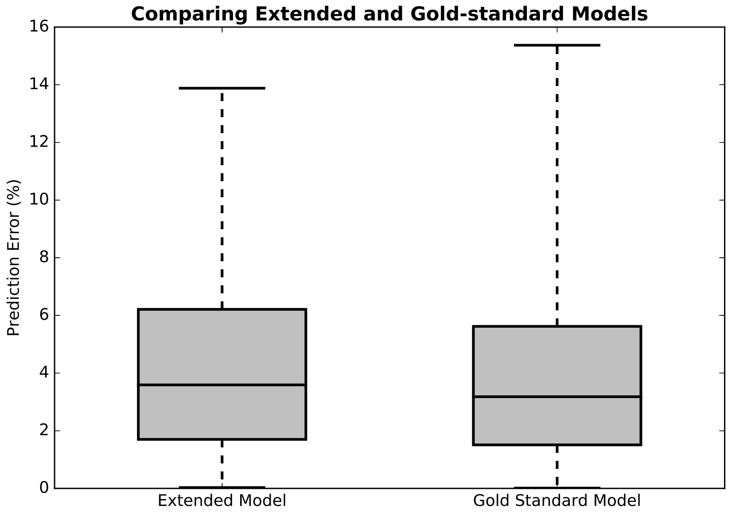

Figure 3.11: Comparing prediction errors achieved by extended and gold of standard models

range, around 0.4%, points out that variation present in both samples are very similar.

Those results confirmed that the extended model really achieved very good predictions for the validation dataset since their results was very close to the ones achieved by the gold standard model. Thus, even being built upon an interference dataset generated from synthetic applications, the extended model was able to perform satisfactory predictions for a set of real HPC applications. This analysis indicates that the model is capable of performing satisfactory out-of-sample predictions.

Chapter 4

Efficient Usage of Physical Machines

The IVMP problem has two objectives: (i) minimizing the interference suffered by applications that share a same physical machine and (ii) minimizing the total number of physical hosts required to allocate those applications in the cloud environment. The first goal aims to rise the performance of HPC applications when executed in clouds, while the second one, besides reducing energy costs, allows cloud provider to rapidly deliver computational resources.

In this chapter, we present the criterion adopted by our proposal to achieve that second objective, i.e., to accomplish an efficient usage of physical resources in such a way that the number of active physical machines can be minimized. First, we briefly discuss the relevance of minimizing the number of used physical machines in clouds. Next, we show that the traditional approach, commonly used to achieve this goal, is not suitable to deal with VMP problem. At last, we describe the two metrics used by our VMP strategy to achieve that goal.

4.1 Background

A typical cloud computing environment is comprised of a set of physical machines, where each of them is equipped with some virtualization mechanism. This virtualization mechanism provides virtual machines where users can execute their own operating system and applications [7]. A physical machine is said to be active, or in use, when it holds, at least, one single virtual machine. Otherwise, that physical machine is considered as being unused or inactive.

There are two main reasons why minimizing the total number of used physical machines in clouds is an important goal to reach.

At first, by minimizing the number of used machines, the cloud provider can promptly satisfy incoming requests because, in such case, there will be, possibly, a greater number of physical machines available to host the new virtual machines. In other words, this goal is essential to provide the illusion of infinite resource capacity offered by clouds. This capacity, though theoretical, allows customers to rapidly obtain computational resources without considering limitations or constraints related to physical space, energy provisioning and cooling capacity. This advantage eliminates the need for users to plan far ahead for provisioning computational resources required to deploy their applications on clouds [52] [9]

Besides providing a rapid provisioning of computational resources, that objective also allows cloud provider to reduce the total energy consumption in datacenter. This happens because an unused physical machine can either be switched off or put in power saving mode, thus reducing the electricity needed to keep datacenter working in a given time interval. As a consequence, cloud providers can decrease its TCO (Total Cost of Ownership) by dwindling energy operational costs. Moreover, as a high percentage of carbon emissions is directly attributed to energy consumption[1], the reduction of power consumption can also contribute to slow down the progress of global climate change [14] [98].

In order to achieve a minimal usage of physical machines, a VMP strategy must accomplish an efficient and balanced usage of physical resources like number of CPUs and amount of main memory. Consider, for example, the two scenarios illustrated in Figure 4.1, where allocation of virtual machines was accomplished in a given time t. In both of those scenarios, namely A and B, the cloud computing environment is comprised of three physical machines, each one endowed with 10 CPUs. In scenario A, after six allocation requests (time *t6*), physical machines M1 and M2 can only host virtual machines with 1 and 2 CPUs, respectively. Thus, to host the virtual machine with 3 CPUS, requested in *t7*, the VMP strategy should necessarily use an addition physical machine, M3. But, if the allocation of virtual machines was accomplished as illustrated in scenario B, that virtual machine with 3 CPUs could be promptly allocated in M1, thus avoiding to activate a new physical machine to satisfy that incoming request. This happened because, in scenario B, placement decisions were made to avoid or reduce resource wastage.

[1]About 98 % of C02 emissions (or 87 % of all CO2-equivalent emissions from all greenhouse gases) can be directly attributed to energy consumption, according to a report by the EIA (Energy Information Administration)

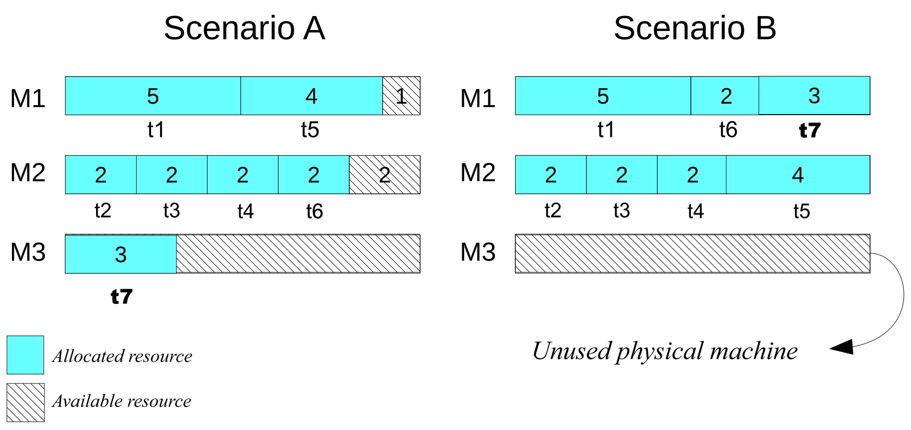

Figure 4.1: VMP

Therefore, as shown by that simple example, the way which virtual machines are allocated in the cloud environment can significantly influence the number of physical machines needed to attend all allocation requests.

4.2 Vector Bin Packing

Several works, like [59] and [68], proposed solutions based on the classical Bin Packing (BP) problem [38] to minimize the number of physical machines in clouds. In this problem, a set of items of various sizes has to be packed into the smallest number of bins [38] [70]. In the context of VMP problem, items and bins are used to represent, respectively, virtual and physical machines. Moreover, dimensions of items and bins correspond, in the VMP problem, to physical resources requested and delivered by virtual and physical machines, respectively. These physical resources can be, for example, the number of CPUs and amount of main memory.

When considering just one dimension (physical resource, in case of VMP problem), both VMP and BP problems are very similar to each other. However, VMP can not be tackled as a BP problem for cases which more than one dimension is being evaluated. Notice that, though two-dimension BP allows to place items side-by-side or one above the other, this is not a valid operation in the context of VMP problem. Such operation can not be executed in VMP context because a virtual machine can not use the same physical resource already assigned to other virtual machine. In other words, once the physical resource is allocated to a virtual machine, it can not be reallocated to another one [76] [87]

To clarify this issue, consider the example illustrated in Figure 4.2. In this example, the Cartesian plane is used to represent the allocation of virtual machines to a given physical machine, where abscissa and ordinate axes correspond to the number of CPUs and amount of main memory, respectively. Note that, in such example, new virtual machines can only be placed in the area indicated by the hatched rectangle. Indeed, this is the unique area which both of these physical resources are not allocated to any virtual machine. In the other hand, if a new virtual machine was placed in areas marked with ”X”, there would be a resource overlapping which means that two virtual machines would use the same physical resource.

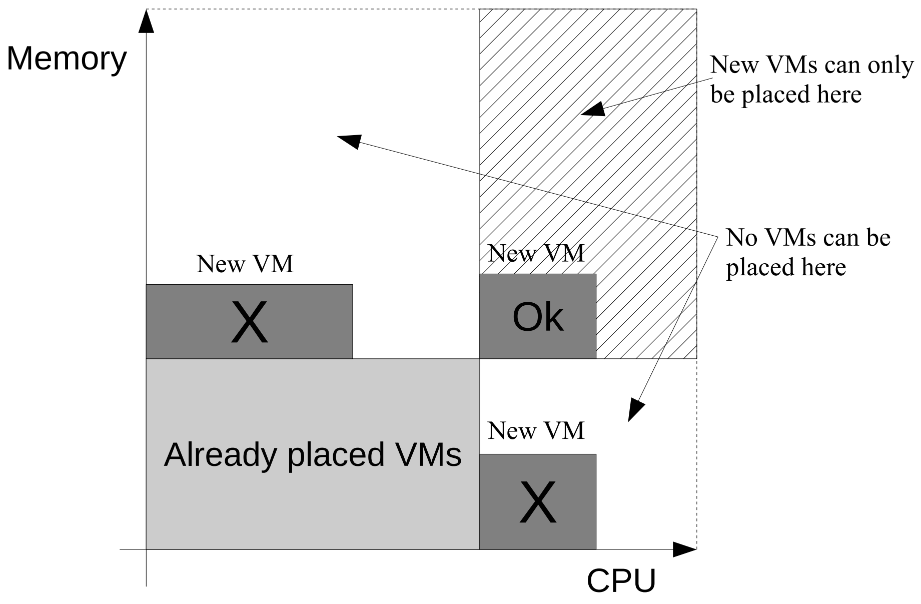

Figure 4.2: VMP in two-dimension resource space [76]

Before this anomaly, some works have used a variant of BP problem to deal with VMP. In this novel approach, called Vector Bin Packing (VBP), available physical resources are represented as multidimensional normalized vectors, where each dimension corresponds to a physical resource. Similarly, requested resources for virtual machines are also represented in the same way. By treating the amount of requested and available resources as a vector, the aforementioned problem of overlapping physical resources is avoided since each physical resource is individually compared and evaluated [76].

4.3 Proposed Criterion

In order to reach a minimal number of physical machines, our VMP strategy adopts a criterion, based on VMP, to select the best physical machine to host a given set of virtual machines. This criterion, which is composed of two metrics, allows to select the physical machine that leads to an efficient and balanced use of physical resources.

These two metrics relies on the following vectors of resources: (i) remaining resources of the physical machine (REM), (ii) requested resources (REQ) and (iii) remaining resources after allocating requested resources (RAF). Each of these vectors has two dimensions, corresponding to the number of CPU and amount of main memory. As the number of CPU (number of cores) and amount of main memory (gigabytes) assumes distinct range of values, we normalized them with respect to the corresponding total amount of resources in the physical machine. For example, a CPU and a memory requests equal to 0.5 means that the application will allocate half of the total amount of CPU and main memory provided in the physical machine.

The first metric, called *alignment factor* (θ), was also used in [43] and is calculated as the angle between vectors REQ and REM, as shown in Equation 4.1. This factor is used to evaluate the shape of the exploitable volume of resources, allowing to assess whether required and remaining resources are complementary to each other. A small alignment factor indicates the required and remaining resources are more complementary to each other. Thus, the VMP strategy should prefer physical machines which would result in a smaller alignment factor in order to accomplish a better usage of physical resources.

$$\theta = acos\left(\frac{(REQ.cpu * REM.cpu) + (REQ.mem * REM.mem)}{(\sqrt{REQ.cpu^2 + REQ.mem^2}) * (\sqrt{REM.cpu^2 + REM.mem^2})}\right) \tag{4.1}$$

To provide a better understanding on this metric, consider, for example, the scenario depicted in Figure 4.3. In such example, a VMP strategy should select, from two physical machines - M1 and M2, the best one to allocate a virtual machine requesting 0.3 and 0.2 of CPU and main memory (vector REQ), respectively. Suppose either that the amount of physical resources, left on each physical machine, is indicated by the values of vector REM. Since both physical machines have enough resources to allocate the new virtual machine, the decision comes down to choosing the one that would yield the smaller resource wastage.

By calculating the alignment factor for both physical machines, we can assert that M1 would be the best choice for that case. As can be seen in Figure 4.3, vectors REM and REQ, in M1, present an smaller angle if compared to the same vectors in M2. This means that the amount of resources required by the new virtual machine fits most to the amount of resources left in M1. Indeed, the incoming virtual machine requires a greater amount of

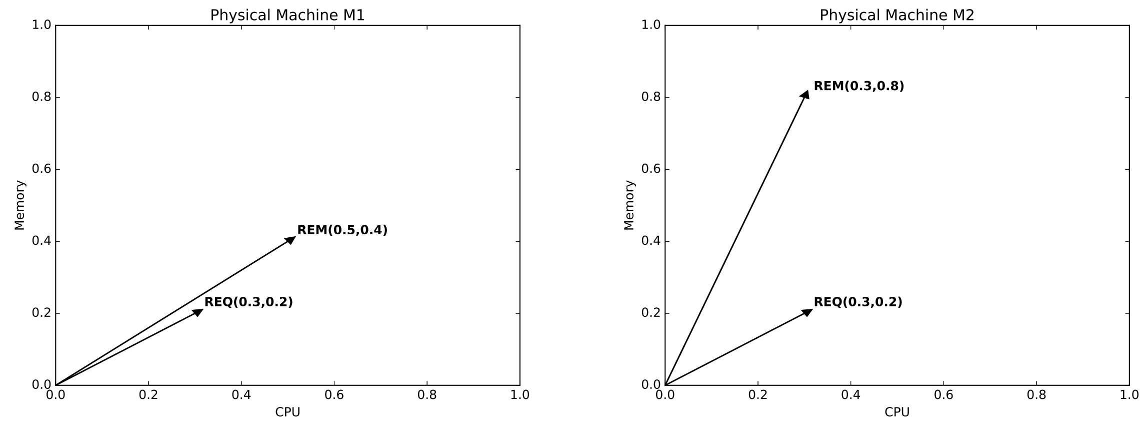

Figure 4.3: Example

CPU than main memory and, in a complementary way, the physical machine M1 presents a greater amount of remaining CPU than main memory. Thus, using M1 to allocate the new virtual machine would imply in a more balanced use of physical resources.

angulo2-eps-converted-to.pdf

Figure 4.4: Recursos disponÃ■veis depois da alocaÃğÃčo de uma mÃąquina virtual

Indeed, as can be seen in Figure X,

Para validar essa escolha, observe na Figura 4.4 como ficaria o uso de recursos caso aquela mÃąquina virtual tivesse sido alocada em M1 e M2. Como pode ser observado, todos os recursos de CPU de M2 estariam esgotados caso a mÃąquina virtual tivesse sido alocada nessa mÃąquina fÃ■sica. EntÃčo, haveria um desperdÃ■cio de mais de 40% de memÃşria, jÃą que nenhuma nova mÃąquina virtual poderia ser alocada em M2. Por outro lado, pode-se observar um uso balanceado de recursos em M1, jÃą que a quantidade de CPUs e memÃşria estÃą sendo consumida na mesma proporÃğÃčo.

Portanto, esse critÃľrio se mostra adequado para realizar uma comparaÃğÃčo entre quais soluÃğÃţes de alocaÃğÃčo permitiriam alcanÃğar uma maximizaÃğÃčo do uso de recursos fÃ■sicos ao minimizar, em contrapartida, o desperdÃ■cio desses recursos.

Although the alignment factor

The other metric, called *residual factor* (λ), is equal to the length of vector RAF and is calculated as shown in Equation 4.2. This metric is used to assess the size of exploitable volume of resources, i.e, it measures the amount of resources left in physical machine after the allocation of the requested resources. This second metric is used as a tiebreaker for the first metric, i.e., for cases with the same alignment factor, the residual factor is used to select the physical machine which will result in the smallest resource wastage.

$$\tau = \sqrt{RAF.cpu^2 + RAF.mem^2} \tag{4.2}$$

We claim that the best physical machine to allocate

Chapter 5

An Interference-aware Virtual Machine Placement Strategy

The Interference-aware Virtual Machine Placement Problem for HPC Applications in Clouds (IVMP) has two objectives: (i) minimize the interference level suffered by HPC applications, and (ii) minimize the number of physical machines used to allocate them in a cloud environment.

Remark that one objective is conflicting with the other one. To minimize the number of used physical machines, applications should be allocated to the smallest possible number of physical machines, resulting in a higher level of interference. On the other hand, cross-interference could be avoided by spreading out applications over distinct physical machines, which would increase the number of used physical machines.

However, even being conflicting, those objectives can be achieved by co-locating applications with complementary access profiles. Therefore, the interference level can be reduced, without increasing the number of used physical machines a lot.

5.1 Mathematical Formulation of IVMP

In this section, we present a mathematical formulation for the IVMP problem whose notations are summarized in Table 5.1. Let M be the set of physical machines available in a cloud environment and A the set of applications to be allocated in that environment. Moreover, we define the total amount of main memory and number of CPUs provided by each physical machine as Mem and Cpu, respectively. Similarly, the amount of CPU and main memory required by an application $i \in A$ is defined as m_i and c_i, respectively. It is worth mentioning that, in this work, we assume that all physical machines have the same hardware configuration, i.e., they are equipped with the same amount of CPU and main memory.

Moreover, we define γ_Q as the interference level experienced by a subset of applications $Q \subseteq A$ when allocated to a same physical machine and ω as the maximum interference level that can be reached on each physical machine. We now define a binary variable $X_{i,j}$ for each $i \in A$ and $j \in M$ such that $X_{i,j}$ is equal to 1 if and only if application i is allocated to the physical machine j, and equal to 0, otherwise. In addition, we define a binary variable Y_j for each $j \in M$ such that Y_j is equal to 1 if and only if the physical machine j is being used, i.e., if there is, at least, one application allocated to it. Y_j is equal to 0 if no application is allocated to j.

$$min \left(\left\{ \frac{\sum_{j \in M} \left[\sum_{Q \subseteq A} \left(\prod_{i \in Q} X_{i,j} \right) .\gamma_Q \right]}{\omega . \sum_{j \in M} Y_j} \right\} .\alpha + \left(\frac{\sum_{j \in M} Y_j}{|M|} \right) .(1 - \alpha) \right) \tag{5.1}$$

$$\sum_{i \in A} X_{i,j} = 1, \forall i \in A \tag{5.2}$$

$$\sum_{i \in A} X_{i,j}.m_i \leq Mem.Y_j, \forall j \in M \tag{5.3}$$

$$\sum_{i \in A} X_{i,j}.c_i \leq Cpu.Y_j, \forall j \in M \tag{5.4}$$

$$X_{i,j} \leq Y_j, \forall i \in A, \forall j \in M \tag{5.5}$$

$$Y_{j+1} \leq Y_j, \forall j = \{1...|M| - 1\} \tag{5.6}$$

$$X_{i,j} \in \{0, 1\}, \forall i \in A, \forall j \in M \tag{5.7}$$

$$Y_j \in \{0, 1\}, \forall j \in M \tag{5.8}$$

This problem can be formulated as the integer programming problem described in

Equations (5.1) to (5.8). The objective function defined in Expression 5.1 seeks for minimization of the sum of interference levels presented in each physical machine and the number of machines used to allocate all applications. Remark that the sum of interference levels was normalized with respect to the maximum sum of interference levels that can be reached in our environment, i.e., $\omega . \sum_{j \in M} Y_j$.

Table 5.1: Description of notations used in mathematical formulation for IVMP

Notation	**Description**
M	Set of available physical machines
A	Set of applications to be allocated
Mem	Total amount of memory on each physical machine
Cpu	Total amount of CPU on each physical machine
m_i	Amount of memory requested by application i
c_i	Amount of CPU requested by application i
γ_Q	Interference level experienced by subset of applications Q
ω	Maximum interference level achieved in the environment
$X_{i,j}$	Variable is equal to 1 whether application i is allocated to machine j
Y_j	Variable is equal to 1 whether physical machine j is being used
α	Weight to determine the relevance of each objective

Besides that, we normalized the number of used machines concerning the total number of physical machines available in the cloud environment. As both objectives were normalized between 0 and 1, we can use the weight α to determine the relevance of each objective. Therefore, when α is set to 1 the objective function prioritizes the minimizing of the sum of interference levels, and when α is close to 0 the function will minimize the number of used physical machines. Thus, this parameter can be adjusted to reach a desirable trade-off between both objectives.

Constraints described in Equation 5.2 ensure that each application is entirely allocated to just one physical machine. Inequalities defined in (5.3) and (5.4) enforce that the total amount of CPU and main memory available in physical machines are not exceeded. In Inequalities (5.5) we defined constraints to guarantee that a physical machine is used if and only if it holds, at least, one single application. As all physical machines permutation yields feasible solutions, we added a set of constraints, defined in (5.6), that eliminates symmetry by establishing an order at which machines shall be used in the cloud environment. At last, constraints described in (5.7) and (5.8) define the binary and integrality requirements on the variables.

5.2 Multistart Iterated Local Search for the IVMP Problem

The classic VMP problem is classified as a NP-Hard problem [86]. So, IVMP, that is a variant of VMP, can also be classified as a NP-Hard problem. For this sort of problem, exact procedures have often proven to be incapable of finding solutions in a reasonable time.

Therefore, we propose in this work a solution based on the Iterated Local Search (ILS) metaheuristic that belongs to the class of local search algorithms such as VNS (Variable Neighborhood Search) and GRASP (Greedy Randomized Adaptive Search Procedure). More specifically, ILS is based on a stochastic search that iteratively applies perturbations on the best solution to generate a random walk in the space of local optima [64]. Moreover, we adopted a multistart approach where new initial solutions are submitted to ILS, allowing it to explore larger areas of the search space.

The input data for our method is comprised of (i) the set of applications to be allocated to the cloud (A), containing for each of them the amount of requested CPU and memory, and the individual access to each of the three shared resources, and (ii) the set of physical machines (M). Notice that the amount of physical resources, CPU and main memory, were normalized with respect to the total amount of resource provided by the physical machine.

In order to present our proposed algorithm, we need to introduce some additional notation. We define a placement solution $s = \{(a1, m1), (a2, m1), (a3, m2)...\}$ as the set of 2-tuples (a, m) representing that the application a is allocated to physical machine m. Moreover, we define $Z(s)$ as the normalized sum of interference levels predicted for solution s. We calculated this sum of interference levels by estimating the level of interference presented by each physical machine of solution s. As early described in Section **??**, we used a quantitative and multivariate interference prediction model to estimate the interference level experienced by applications allocated to a single physical machine.

In addition, we define $M(s)$ as the normalized number of physical machines used in s, and $E_{CPU}(s)$ and $E_{MEM}(s)$ as the percentage of amount of CPU and main memory exceeded in solution s, respectively. A solution is considered feasible if and only if $E_{CPU}(s)$ and $E_{MEM}(s)$ are equal to zero.

We define in Equation (5.9) a cost function $f : \mathbb{S} \rightarrow \mathbb{R}$, where $\mathbb{S}$ is the set of all possible solutions. Similar to (5.1), this function $f(s)$ attempts to minimize the sum of levels of interference and the number of used machines. In addition, $f(s)$ penalizes, in accordance with parameter λ, unfeasible solutions, which use more CPU and memory than the available amount in the environment.

$$f(s) = Z(s).\alpha + M(s).(1 - \alpha) + E_{CPU}(s).\lambda + E_{MEM}(s).\lambda \tag{5.9}$$

Our proposed solution, called *ILSivmp*, is described in Algorithm 2. For each iteration of the *Multistart Loop* (lines 2 to 15), the algorithm executes the procedure *ConstructivePhase* (line 3) to create a new solution to be submitted to the ILS. Then, *ILS Loop* (lines 4 to 13) performs a local search in the neighborhood of the current solution through *Variable Neighborhood Descent* (VND) method (line 5). Both VND and *ConstructivePhase* are described in details later.

After executing VND, the *ILSivmp* evaluates the quality of the current solution $\overline{s}$ (line 6). If $\overline{s}$ represents an improvement on the best current solution s^*, it is set as the actual best solution and the counter j is reset to 1 (lines 7 and 8). Otherwise, the counter j is incremented (line 10). Then, the algorithm executes the procedure *Perturbation* (line 12) that applies a perturbation on the current solution in order to escape from local optima.

Perturbation executes j random movements of three classical local search movements for the original VMP problem: *Move1*, *Move2* and *Swap1*. Heuristics *Move1* and *Move2*, move, respectively, one and two applications to another used physical machine, while *Swap1* performs one swap operation, i.e., it swaps two applications between two distinct physical machines. Furthermore, this procedure also adds a new physical machine to the solution, before executing the last perturbation (i.e., when j is equal to $iterMaxILS$) of a given initial solution (i.e., solution created by *ConstructivePhase*).

Algorithm 3 presents the *ConstructivePhase* which aims to rapidly generate high-quality initial solutions. Firstly, the algorithm sorts the list of applications in decreasing order (line 1) by considering the following sequence: (i) amount of accesses to SLLC, (ii) amount of requested CPU and main memory, and (iii) amount of accesses to virtual network and DRAM. Next, the algorithm inserts one physical machine into the set of used machines (lines 2 and 3). Then, the procedure executes *Allocation Loop* (lines 4 to 20) that basically performs the selection of two applications from the list of sorted applications and selection of a physical machine to allocate them.

The process of selecting this pair of applications is described as follows. Firstly, the algorithm creates the sets of applications $First$ and $Last$ that are composed, respectively,

Algorithm 2 *ILSivmp*

```
    Input: $A$,$M$,$\beta$,$iterMaxILS$, $iterMaxMultiStart$
    Output: $s^*$
 1: $s^* = \emptyset$; $f(s^*) = \infty$; $i = 1$; $j = 1$;
    /*Multistart Loop*/
 2: while $i \leq iterMaxMultiStart$ do
 3:     $s = ConstructivePhase(A, M, \beta)$;
        /*ILS Loop*/
 4:     repeat
 5:         $\overline{s} = VND(s, A, M)$;
 6:         if $(f(\overline{s}) < f(s^*))$ then
 7:             $s^* = \overline{s}$;
 8:             $j = 1$;
 9:         else
10:             $j = j + 1$;
11:         end if
12:         $s = Perturbation(\overline{s}, j)$;
13:     until $j \leq iterMaxILS$
14:     $i = i + 1$;
15: end while
16: return $s^*$
```

of the first and last N_a elements of the sorted list of applications (lines 5 to 7). This number of applications is defined by the parameter β that is used to control the degree of randomness of this constructive phase. Next, the algorithm selects, randomly, applications i and j from $First$ and $Last$, respectively (line 8). By selecting applications from the head and tail of the sorted list, this process aims to co-locate applications with complementary access profiles and amount of requested resources, thus resulting in a low interference level.

After selecting this pair of applications, the best physical machine to allocate them is chosen among the set of physical machines with enough resources to allocate both applications. The method selects the best physical machine by determining the one with the smallest alignment and residual factors (in this order) to allocate applications i and j (line 10).

However, if none of the used physical machines has available resources to allocate applications i and j, the *ConstructivePhase* uses a new physical machine from M to allocate them (lines 12 to 14). This approach aims to reduce the number of used machines because it takes new machines only when it is absolutely necessary. Moreover, if none of physical machines is capable of allocating that pair of applications, the method chooses randomly a physical machine to hold them (line 16). At last, both applications i and j are

Algorithm 3 *ConstructivePhase*

```
    Input: $A$,$M$,$\beta$
    Output: $s$
 1: $App$ = $Sort(A)$;
 2: $Used = Used \cup \{m_1\}$;
 3: $M = M \backslash \{m_1\}$;
    /*Allocation Loop*/
 4: while $App \neq \emptyset$ do
 5:     $N_a = \lceil \beta.|App| \rceil$;
 6:     $First$ = first $N_a$ elements of $App$;
 7:     $Last$ = last $N_a$ elements of $App$;
 8:     Choose randomly applications $i \in First$ and $j \in Last$;
 9:     if There is $m \in Used$ with enough resources to allocate $i$ and $j$ then
10:         Choose the best one to allocate $i$ and $j$;
11:     else if $M \neq \emptyset$ then
12:         Pick up any $m \in M$
13:         $Used = Used \cup \{m\}$;
14:         $M = M \backslash \{m\}$;
15:     else
16:         Choose randomly $m \in Used$;
17:     end if
18:     $s = s \cup \{(i,m),(j,m)\}$;
19:     $App = App \backslash \{i,j\}$;
20: end while
21: return $s$
```

allocated to the selected physical machine and removed from the sorted list of applications (lines 18 and 19). Notice that unfeasible solutions can be generated by this constructive method. However, their corresponding costs are very high due to the penalty applied in the proposed cost function.

Algorithm 4 presents the VND method. This method executes iteratively those classical local search movements for the original VMP problem, that is, it executes the *Move1*, *Swap1*, and *Move2*, in this order. As we adopted the strategy of first improving, these heuristics stop executing upon improving the current solution. In addition, the VND halts when no improvement is achieved by any of these three heuristics.

5.3 Experimental Tests and Results

We carried out experiments to assess our proposal concerning its ability for reducing the sum of interference levels, while reducing also the number of used machines. Note that, in our previous work [6], we showed that the proposed prediction model was able to

Algorithm 4 *VND*

```
    Input: $s$,$A$,$M$
    Output: $s^*$
 1: $Mov1 = Swa1 = Mov2 =$ True;
 2: while $Mov1$ or $Swa1$ or $Mov2$ do
 3:     $Mov1 = Swa1 = Mov2 =$ False;
 4:
 5:     $s' = Move1(s, A, M)$;
 6:     if $(f(s') < f(s))$ then
 7:         $s = s'$;
 8:         $Mov1 =$ True;
 9:     end if
10:
11:     if (not Mov1) then
12:         $s' = Swap1(s, A, M)$;
13:         if $(f(s') < f(s))$ then
14:             $s = s'$;
15:             $Swa1 =$ True;
16:         end if
17:     end if
18:
19:     if (not Mov1 and not Swa1) then
20:         $s' = Move2(s, A, M)$;
21:         if $(f(s') < f(s))$ then
22:             $s = s'$;
23:             $Mov2 =$ True;
24:         end if
25:     end if
26: end while
27: $s^* = s$;
28: return $s^*$
```

determine the interference level with a reasonable prediction error. In this work, we expect that the use of this prediction model in our proposed VMP strategy does not sacrifice other important aspect that should be considered when solving the VMP problem: the efficient usage of physical resources.

Thus, we compared our VMP strategy with some of the most widely employed heuristics to minimize the number of used machines when allocating virtual machines in clouds: Best Fit (BF), First Fit (FF), Worst Fit (WF), Best Fit Decreasing (BFD), First Fit Decreasing (FFD) and Worst Fit Decreasing (WFD). Since these greedy algorithms have shown to be effective for solving this problem in previous works [33] [11], they were also used as a baseline here, in our experiments.

Experimental tests were conducted in two phases. At first, we performed several offline experiments to calibrate the value of parameter α, i.e., to find out the α that gives a good trade-off between reducing both interference and the number of used physical machines. After that, we executed experiments in a real scenario, using the calculated α.

Before presenting these results, we briefly describe the real applications used in our experiments.

5.3.1 Calibrating Parameter Alpha

Before evaluating our proposal in a real scenario, we conducted offline experiments to calibrate the input parameter α. This parameter is used to adjust the behavior of *ILSivmp* so that it can minimize, as much as possible, both objectives.

In order to execute such experiment, we created *test cases* for the IVMP problem by considering the set of applications described in the previous section. A test case for the IVMP problem is composed of (i) the set of applications to be allocated in the cloud environment and (ii) the number of physical machines available in that environment. For each application, the test case must provide the amount of individual access to SLLC, DRAM and virtual network, and the normalized amount of requested CPU and main memory.

We created 100 test cases for the IVMP problem, where the number of applications at each test case varied from 5 to 50 (interval of 5). Each test case was generated by selecting randomly a number of applications from the workload described in the previous section. Notice that an application can appear more than once in a test case. In addition, the number of machines available in the cloud environment was set as the total number of applications to be allocated.

We tested our proposal and the other heuristics over that set of test cases. In order to assess our proposal against those heuristics, we devised the following two metrics. In the first one, we determined the percentage of test cases where our solution achieved a *strictly smaller sum of interference levels* than the one reached by the heuristic. Concerning the minimization of interference, this metric allows to quantify the number of test cases where our solution outperformed the tested heuristic.

In the second metric, we calculated the percentage of cases where our proposal used a *smaller or equal number of physical machines* than the one used by the heuristic. By using this metric, we expect to evaluate the number of cases where our solution achieved the minimal number of physical machines needed to allocate all applications in the cloud

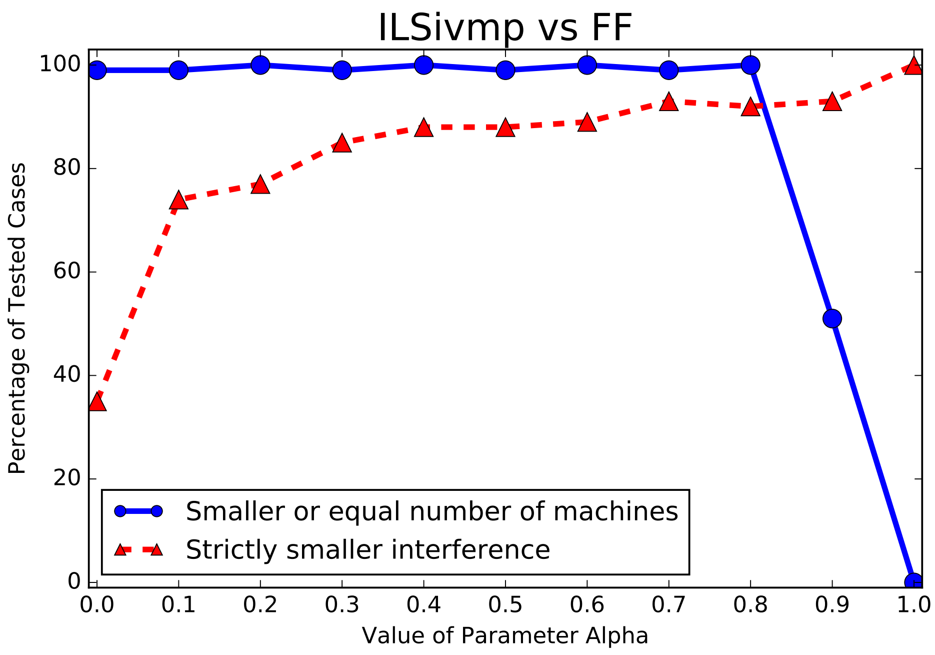

Figure 5.1: ILSvmp vs FF

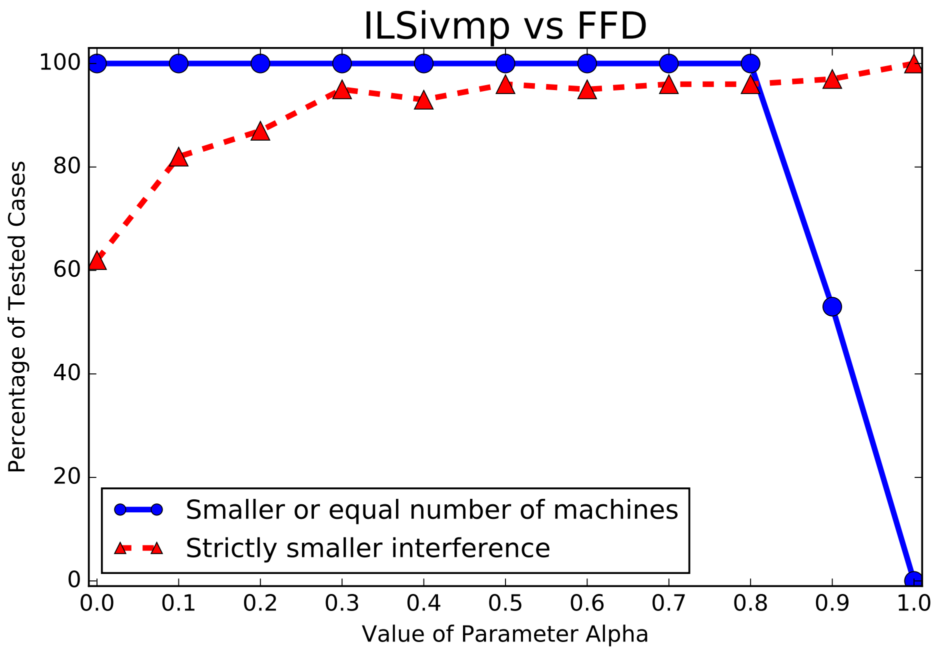

Figure 5.2: ILSvmp vs FF

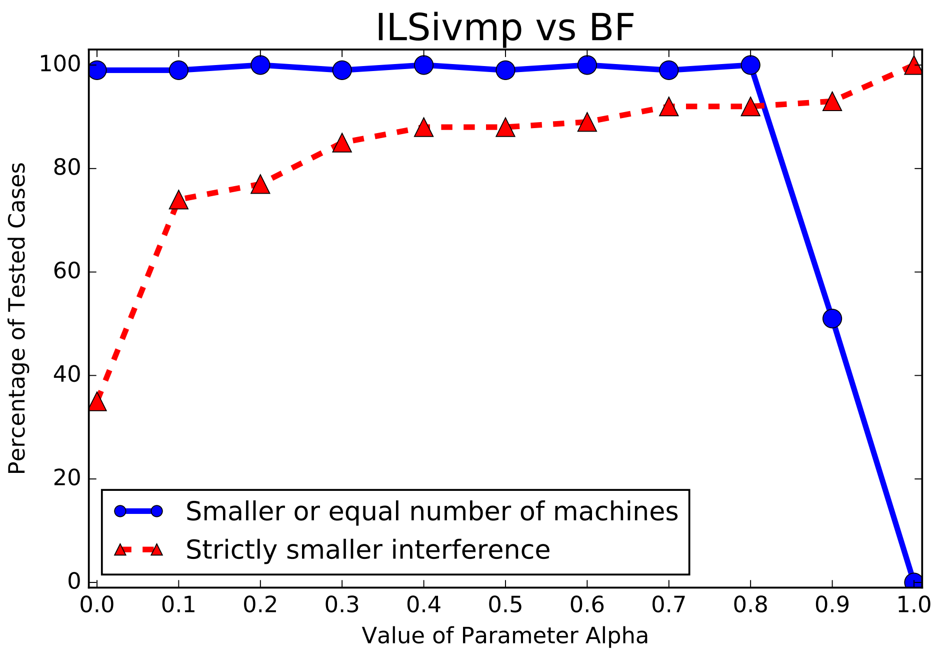

Figure 5.3: ILSvmp vs FF

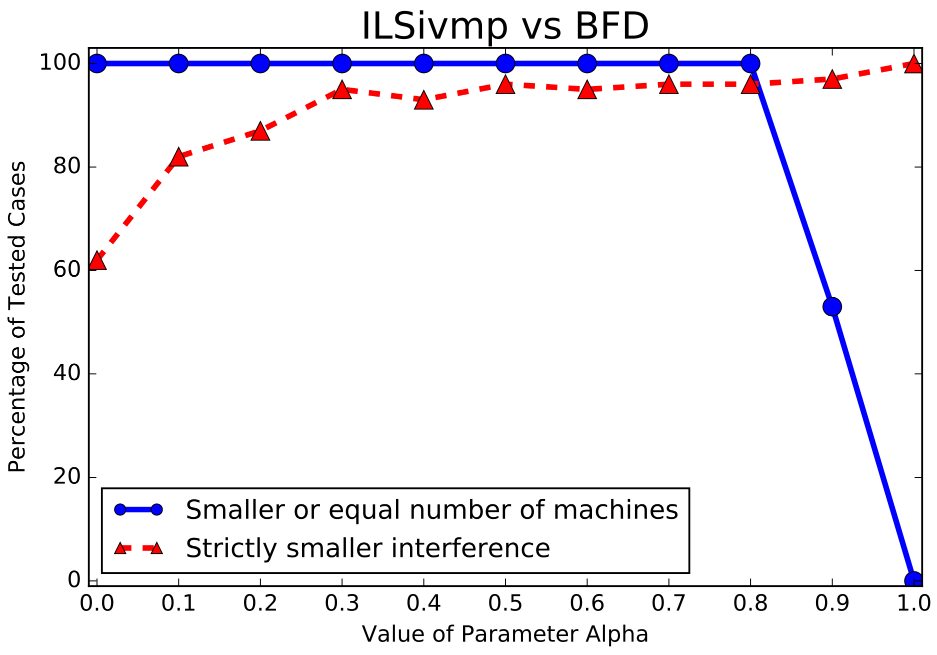

Figure 5.4: ILSvmp vs FF

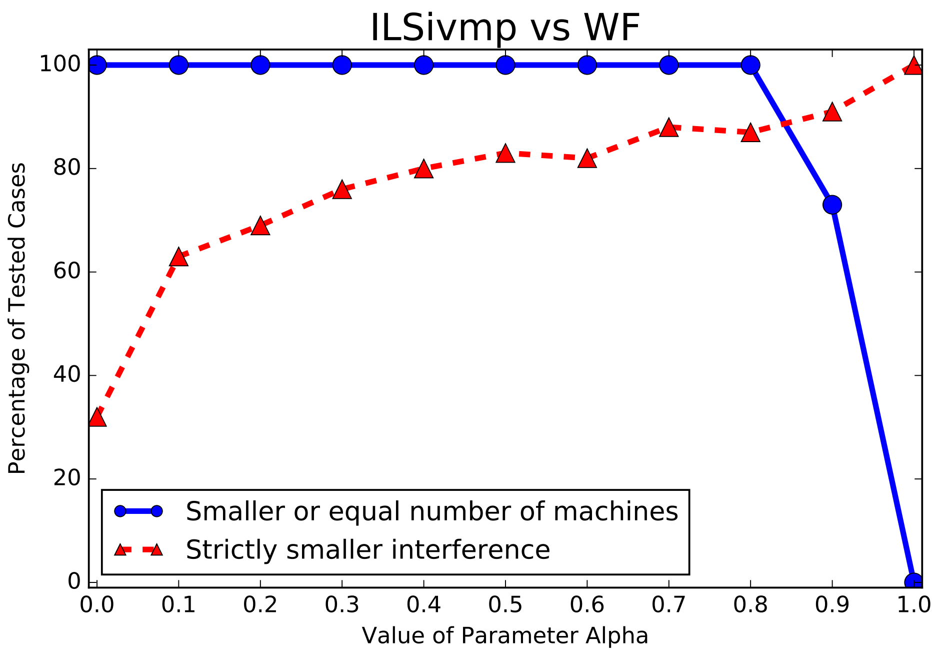

Figure 5.5: ILSvmp vs FF

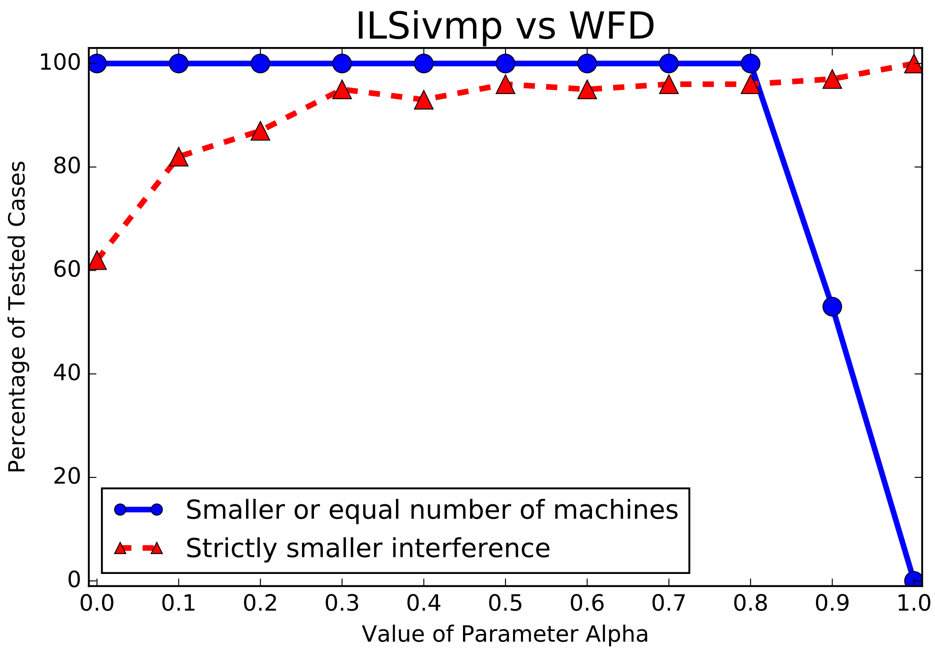

Figure 5.6: ILSvmp vs FF

environment.

Remark that, when comparing with a single heuristic, the best trade-off between minimizing interference and the number of physical machines is achieved when both of those metrics reach, simultaneously, 100% of test cases. In this hypothetical scenario, the *ILSivmp* would reduce interference for all test cases, while keeping, at least, the same number of used physical machines given by that heuristic.

Concerning ILS, α was varied from 0.00 to 1.00 (interval of 0.10), while parameters β, λ, *iterMaxILS* and *iterMaxMultistart* were fixed in 0.4, 0.5, 10 and 50, respectively. As *ILSivmp* is a non-deterministic algorithm, we repeated the execution of each test case until reaching a coefficient of variation[1] smaller than 2%. Moreover, the longest execution time of *ILSivmp* in all test cases was less than 15 seconds. At last, all codes were compiled with GCC (Gnu C Compiler) version 4.9.2 with optimization flag *O3* which enables the process of automatic vectorization [69].

Results of the comparison between *ILSivmp* and heuristics FF, BF, WF, FFD, BFD, and WFD are presented in Figure **??**. As expected, when α was set to 0.00 or 1.00, the *ILSivmp* prioritizes one or another objective. More specifically, when α was equal to 1.0, the *ILSivmp* achieved, in comparison with all heuristics, a strictly smaller sum of interference levels in 100% of test cases, though using a greater number of physical machines in all of those cases. This happens because, in such configuration, the *ILSivmp* allocates each application on each physical machine available in the cloud environment what consequently increases the number of used physical machines.

On the other hand, when α was set to 0.00, the *ILSivmp* presented, for all heuristics, the worst result concerning interference minimization. However, even in that case, our solution outperformed heuristics in, at least, 32% of test cases (result for WF). This behaviour can be explained by the strategy used to construct initial solutions. As previously described in Section 5.2, an initial solution is generated by co-allocating applications with complementary access profiles and complementary amount of requested resources. As a consequence, the initial solution tends to present a low sum of interference levels and number of physical machines, as well. Thus, even when *ILSivmp* seeks only for the minimization of used physical machines, the initial solution may already have a low sum of interference levels.

Considering all experiments, the best trade-off between reducing interference and

[1]Also known as relative standard deviation, the coefficient of variation is defined as the ratio of the standard deviation to the mean.

number of used machines was achieved by our proposal when α was equal to 0.70. In this case, our proposal has achieved a smaller or equal number of used machines in 100% of test cases for all heuristics, besides reaching a smaller interference in 88% of test cases as shown in Figure 5.5. In other words, this value of α enabled *ILSivmp* to reach, at the same time, the higher values for both metrics in all experiments.

However, it is worth mentioning that our solution did not achieve a strictly smaller interference in 100% of test cases in any experiment. In other words, for some of test cases, the *ILSivmp* presented a greater or equal sum of interference levels than heuristics, as can be seen in Figure 5.7. In some cases, the *ILSivmp* achieved greater interference because the heuristic used a greater number of physical machines what naturally reduced the sum of interference levels. However, in only 1% of test cases, the heuristic indeed outperformed our solution by achieving a smaller sum of interference levels with the same number of machines. Even so, the difference between the sum of interference levels achieved by our solution and the one given by heuristics was smaller than 1%.

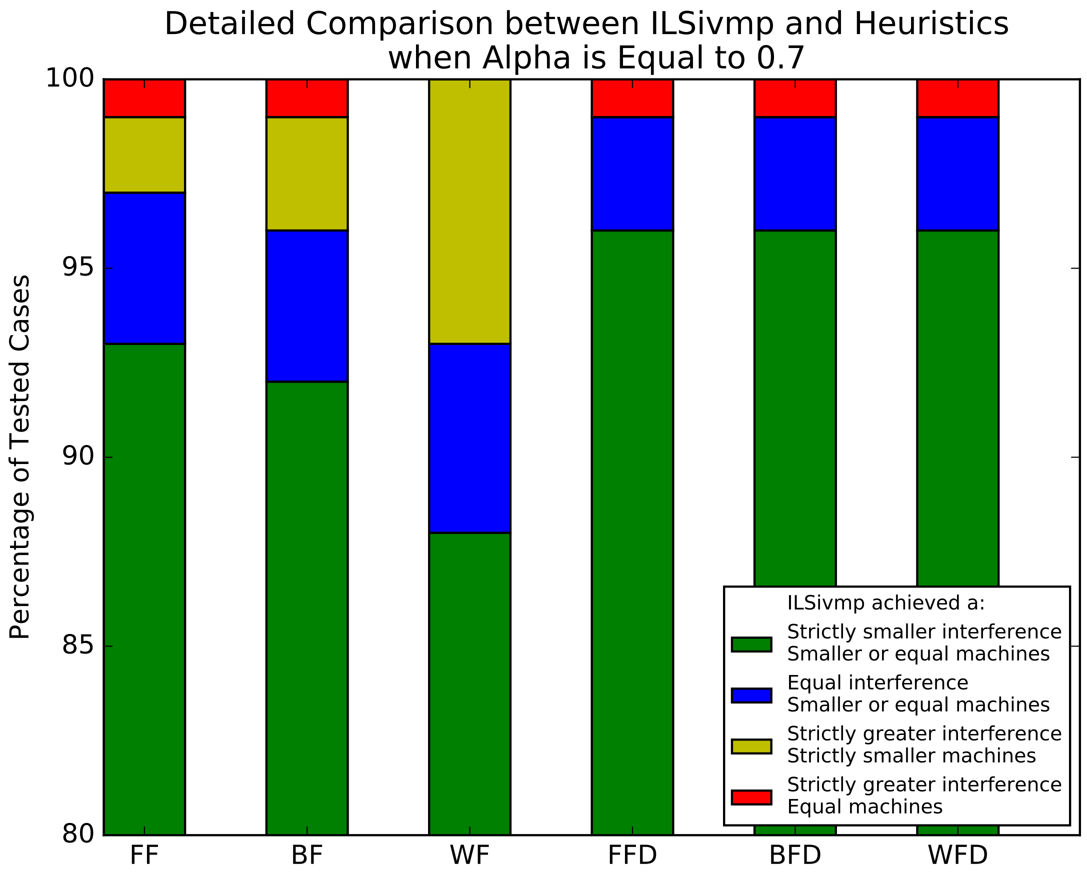

Figure 5.7: Comparing in details results achieved by ILSivmp and heuristcs when alpha=0.7

Those results indicate that our proposal was able to reduce the sum of interference levels, while using a small, perhaps minimal, number of physical machines. In addition, while not primary goal of our proposal, results showed that the cloud provider can use our strategy to dynamically adjust the behavior of its virtual machine placement policy by varying the value of parameter α. Thus, the cloud provider can easily prioritizes one or another objective depending on its needs.

5.3.2 Experiments in a Real Scenario

By using that selected value of α, we conducted a second phase of experiments to verify, in a real scenario, the efficiency of *ILSivmp* in reducing the sum of interference levels, while keeping the same number of physical machines given by the greedy heuristics.

Thereby, we selected five test cases where the number of physical machines given by our solution and heuristics was exactly the same. In this way, we could accomplish a fair comparison between the tested strategies. For each test case, we executed applications in accordance with the placement determined by the *ILSivmp* and by the heuristic that presented the lower sum of interference levels in offline experiments. Next, we calculated the sum of interference levels that our proposal and the heuristic achieved in the real scenario. These experiments were executed in a set of physical machines whose configurations were detailed in Section 3.2.2.

Results show that our strategy, even using the same number of physical machines of the other heuristic, reduced the sum of interference levels in more than 40%, as presented in Figure 5.8. Remark that, for the test case E, our strategy achieved a sum of interference levels 41% smaller than the one reached by the best heuristic, FF in this case. It is also worth mentioning that the difference between the predicted sum of interference levels and the one achieved in real experiments was around 8%.

Those results allowed to confirm conclusions drawn from offline experiments. The virtual machine placement indicated by the *ILSivmp* was able to reduce the level of interference experienced by HPC applications without increasing the number of physical machines needed to allocate them in the cloud environment.

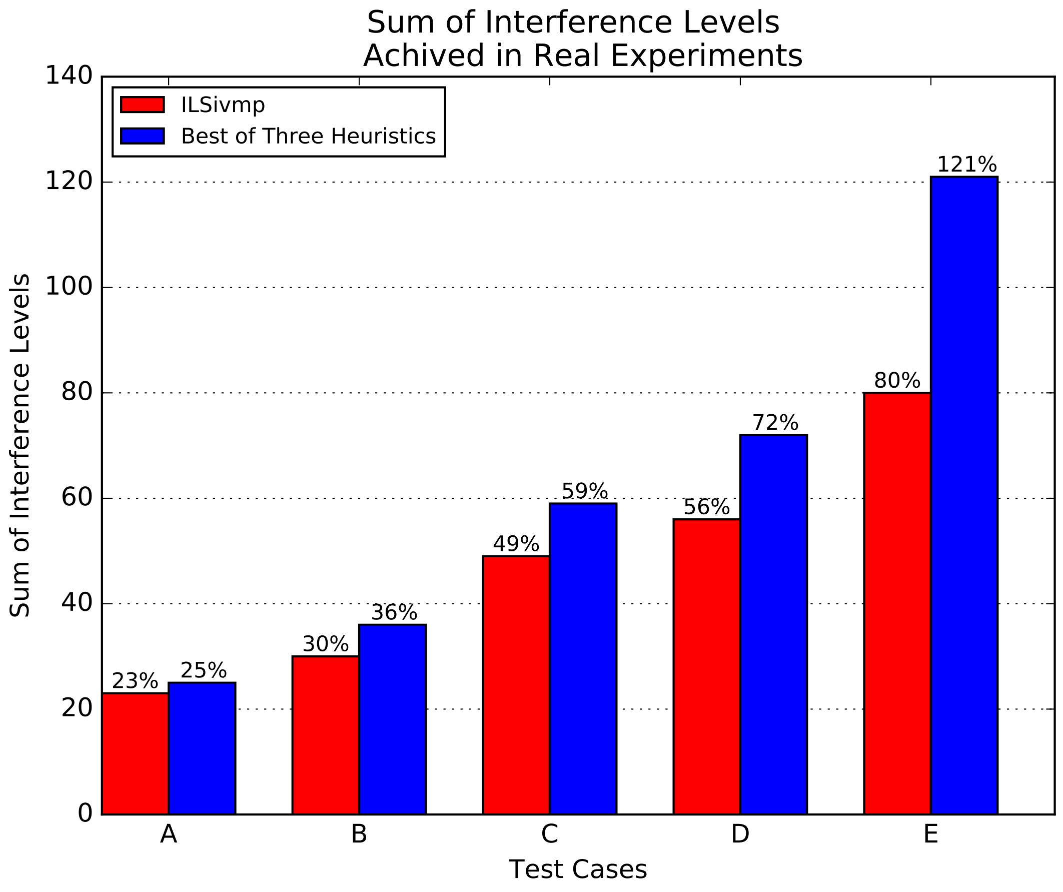

Figure 5.8: Comparing sum of interference levels in a real scenario

5.4 Conclusion and Future Work

In this work, we defined the Interference-aware Virtual Machine Placement Problem for HPC applications (IVMP) that aims to minimize, simultaneously, (i) the interference experienced by HPC applications executed in a same physical machine and (ii) the number of physical machines used to allocate them. Besides presenting a mathematical formulation for the problem, we introduced a stategy based on the Iterated Local Search (ILS) framework to solve it. In order to predict the interference, we extended a quantitative and multivariate model so that it considers a varying number of applications allocated to a same physical machine. Experiments conducted in a real scenario showed that our proposed strategy reduced interference in more than 40% keeping the same number of physical machines when compared to the most employed heuristics from related literature. As future work, we expect to evaluate the influence that concurrent access to other shared resources, like disk, may have in the IVMP.

Chapter 6

Conclusion

Trabalhos futuros: juntar sugestÃţes de TR do JSS e do IPDPS

APPENDIX A – Published Papers

- WSCAD 2014
- WSCAD 2015
- ERAD 2016
- Concurrency and Computation 2016
- Journal of Systems and Software 2017

Bibliography

[1] Abdelkhalek, R.; Calandra, H.; Coulaud, O.; Roman, J.; Latu, G. Fast seismic modeling and reverse time migration on a gpu cluster. In *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on* (2009), IEEE, pp. 36–43.

[2] Afanasyev, A.; Kempka, T.; Kühn, M.; Melnik, O. Validation of the mufits reservoir simulator against standard industrial simulation tools for co2 storage at the ketzin pilot site. In *EGU General Assembly Conference Abstracts* (2016), vol. 18, p. 6883.

[3] Albericio, J.; Ibáñez, P.; Viñals, V.; Llabería, J. M. The Reuse Cache: Downsizing the Shared Last-level Cache. In *Proceedings of the 46th Annual International Symposium on Microarchitecture* (2013), ACM, pp. 310–321.

[4] Alves, M.; Drummond, L. Análise de Desempenho de um Simulador de Reservatórios de Petróleo em um Ambiente de Computação em Nuvem. In *XV Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD 2014) - Brazil* (2014).

[5] Alves, M.; Pestana, R.; Silva, R.; Drummond, L. Accelerating pre-stack kirchhoff time migration by manual vectorization. *Concurrency and Computation: Practice and Experience (online)* (2016).

[6] Alves, M. M.; de Assumpção Drummond, L. M. A Multivariate and Quantitative Model for Predicting Cross-application Interference in Virtual Environments. *Journal of Systems and Software 128* (2017), 150–163.

[7] Andreolini, M.; Casolari, S.; Colajanni, M.; Messori, M. Dynamic load management of virtual machines in cloud architectures. In *International Conference on Cloud Computing* (2009), Springer, pp. 201–214.

[8] Anzt, H.; Dongarra, J.; Gates, M.; Kurzak, J.; Luszczek, P.; Tomov, S.; Yamazaki, I. Bringing high performance computing to big data algorithms. In *Handbook of Big Data Technologies*. Springer, 2017, pp. 777–806.

[9] Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R. H.; Konwinski, A.; Lee, G.; Patterson, D. A.; Rabkin, A.; Stoica, I., et al. Above the clouds: A berkeley view of cloud computing. Tech. rep., Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[10] Atif, M.; Kobayashi, R.; Menadue, B. J.; Lin, C. Y.; Sanderson, M.; Williams, A. Breaking hpc barriers with the 56gbe cloud. *Procedia Computer Science 93* (2016), 3–11.

[11] Babu, K. R.; Samuel, P. Virtual Machine Placement for Improved Quality in IaaS Cloud. In *Fourth International Conference on Advances in Computing and Communications (ICACC)* (2014), IEEE, pp. 190–194.

[12] Baiocchi, G.; Distaso, W. Gretl: Econometric software for the gnu generation. *Journal of Applied Econometrics 18*, 1 (2003), 105–110.

[13] Basto, D. T. Interference Aware Scheduling for Cloud Computing. Master's thesis, Universidade do Porto, 2015.

[14] Beloglazov, A.; Abawajy, J.; Buyya, R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems 28*, 5 (2012), 755–768.

[15] Benedict, S.; Rejitha, R.; Preethi, C.; Bright, C. B.; Judyfer, W. Energy Analysis of Code Regions of HPC Applications using Energy Analyzer Tool. *International Journal of Computational Science and Engineering 14*, 3 (2017), 267–278.

[16] Bhardwaj, D.; Phadke, S.; Yerneni, S. On improving performance of migration algorithms using mpi and mpi-io. In *Expanded Abstracts, Society of Exploration Geophysicists* (2000).

[17] Bobroff, N.; Kochut, A.; Beaty, K. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on* (2007), IEEE, pp. 119–128.

[18] Brown, K.; Xu, T.; Iwabuchi, K.; Sato, K.; Moody, A.; Mohror, K.; Jain, N.; Bhatele, A.; Schulz, M.; Pearce, R., et al. Accelerating big

data infrastructure and applications (ongoing collaboration). In *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on* (2017), IEEE, pp. 343–347.

[19] Chen, L.; Patel, S.; Shen, H.; Zhou, Z. Profiling and Understanding Virtualization Overhead in Cloud. In *44th International Conference on Parallel Processing (ICPP)* (2015), IEEE, pp. 31–40.

[20] Chen, L.; Shen, H.; Platt, S. Cache Contention Aware Virtual Machine Placement and Migration in Cloud Datacenters. In *24th International Conference on Network Protocols (ICNP)* (2016), IEEE, pp. 1–10.

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

[22] Chiang, M.; Yang, C.; Tu, S. Kernel Mechanisms with Dynamic Task-aware Scheduling to Reduce Resource Contention in NUMA Multi-core Systems. *Journal of Systems and Software 121* (2016), 72–87.

[23] Chong, A. Y. L.; ChâĂŹng, E.; Liu, M. J.; Li, B. Predicting consumer product demands via big data: the roles of online promotional marketing and online reviews. *International Journal of Production Research 55*, 17 (2017), 5142–5156.

[24] Coco, A.; Gottsmann, J.; Whitaker, F.; Rust, A.; Currenti, G.; Jasim, A.; Bunney, S. Numerical models for ground deformation and gravity changes during volcanic unrest: simulating the hydrothermal system dynamics of a restless caldera. *Solid Earth 7*, 2 (2016), 557.

[25] da Silva, R. A. P.; Alves, M. M.; Bentes, C. B.; de Assumpção Drummond, L. M. Vetorização e análise de algoritmos paralelos para a migração kirchhoff pré-empilhamento em tempo. *XVIII Simpósio em Sistemas Computacionais de Alto Desempenho-WSCAD* (2017).

[26] Dali, N.; Bouamama, S. Different parallelism levels using gpu for solving max-csps with pso. In *Evolutionary Computation (CEC), 2017 IEEE Congress on* (2017), IEEE, pp. 2070–2077.

[27] de Carvalho Junior, F. H.; Tuyttens, D. A gpu-based backtracking algorithm for permutation combinatorial problems. In *Algorithms and Architectures for*

Parallel Processing: 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings (2016), vol. 10048, Springer, p. 310.

[28] Dehury, C. K.; Sahoo, P. K. Design and Implementation of a Novel Service Management Framework for IoT Devices in Cloud. *Journal of Systems and Software 119* (2016), 149–161.

[29] Dongarra, J.; Luszczek, P. Hpc challenge: Design, history, and implementation highlights. *Contemporary High Performance Computing: From Petascale Toward Exascale* (2013).

[30] Dongarra, J. J.; Luszczek, P. Introduction to the hpcchallenge benchmark suite. Tech. rep., DTIC Document, 2004.

[31] El-Gazzar, R.; Hustad, E.; Olsen, D. H. Understanding Cloud Computing Adoption Issues: A Delphi Study Approach. *Journal of Systems and Software 118* (2016), 64–84.

[32] Expósito, R. R.; Taboada, G. L.; Ramos, S.; Touriño, J.; Doallo, R. Performance analysis of hpc applications in the cloud. *Future Generation Computer Systems 29*, 1 (2013), 218–229.

[33] Fang, S.; Kanagavelu, R.; Lee, B.-S.; Foh, C. H.; Aung, K. M. M. Power-efficient Virtual Machine Placement and Migration in Data Centers. In *Green Computing and Communications (GreenCom)* (2013), IEEE, pp. 1408–1413.

[34] Filgueira, R.; Carretero, J.; Singh, D. E.; Calderón, A.; Núñez, A. Dynamic-CoMPI: Dynamic Optimization Techniques for MPI Parallel Applications. *The Journal of Supercomputing 59*, 1 (2012), 361–391.

[35] Gentzsch, W.; Yenier, B. The uber-cloud experiment. *HPCwire, June 28* (2012).

[36] Gentzsch, W.; Yenier, B. The ubercloud hpc experiment: compendium of case studies. Tech. rep., Tabor Communications, 2013.

[37] Gholami, M. F.; Daneshgar, F.; Low, G.; Beydoun, G. Cloud migration processâĂŤa survey, evaluation framework, and open challenges. *Journal of Systems and Software 120* (2016), 31–69.

[38] Goldbarg, M. C.; Luna, H. P. L. *Otimização Combinatória e Programação Linear: Modelos e Algoritmos*, vol. 1. Editora Campus, Rio de Janeiro, 2000.

[39] Green, S. B.; Salkind, N. J. *Using SPSS for Windows and Macintosh: Analyzing and understanding data.* Prentice Hall Press, 2010.

[40] Guan, W.; Qiao, C.; Zhang, H.; Zhang, C.-S.; Zhi, M.; Zhu, Z.; Zheng, Z.; Ye, W.; Zhang, Y.; Hu, X., et al. On robust and efficient parallel reservoir simulation on tianhe-2. In *SPE Reservoir Characterisation and Simulation Conference and Exhibition* (2015), Society of Petroleum Engineers.

[41] Gupta, A. *Techniques for efficient high performance computing in the cloud.* PhD thesis, University of Illinois, 2014.

[42] Gupta, A.; Faraboschi, P.; Gioachin, F.; Kale, L. V.; Kaufmann, R.; Lee, B.-s.; March, V.; Milojicic, D.; Suen, C. H. Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud. *IEEE Transactions on Cloud Computing 7161* (2014), 1–1.

[43] Gupta, A.; Kale, L. V.; Milojicic, D.; Faraboschi, P.; Balle, S. M. HPC-aware VM Placement in Infrastructure Clouds. In *International Conference on Cloud Engineering (IC2E)* (2013), IEEE, pp. 11–20.

[44] Gupta, A.; Milojicic, D. Evaluation of HPC Applications on Cloud. In *Open Cirrus Summit (OCS)* (2011), IEEE, pp. 22–26.

[45] Habiballah, W.; Hayder, M.; Khan, M.; Issa, K.; Zahrani, S.; Shaikh, R.; Uwaiyedh, A.; Tyraskis, T.; Baddourah, M., et al. Parallel reservoir simulation utilizing pc-clusters in massive reservoir simulation models. In *SPE Annual Technical Conference and Exhibition* (2003), Society of Petroleum Engineers.

[46] Hair, J. F.; Black, W. C.; Babin, B. J.; Anderson, R. E.; Tatham, R. L. *Multivariate Data Analysis (Vol. 6).* Upper Saddle River, NJ: Pearson Prentice Hall, 2006.

[47] Hamid, N. A. W. A.; Coddington, P. Comparison of mpi benchmark programs on shared memory and distributed memory machines (point-to-point communication). *The International Journal of High Performance Computing Applications 24*, 4 (2010), 469–483.

[48] Hassan, H. A.; Mohamed, S. A.; Sheta, W. M. Scalability and communication performance of hpc on azure cloud. *Egyptian Informatics Journal 17*, 2 (2016), 175–182.

[49] Hayder, M. E.; Baddourah, M.; Harbi, B.; Al-Zawawi, A. S.; Abouheit, F.; Zamil, K.; Nahdi, U., et al. Designing a high performance computational platform for simulation of giant reservoir models. In *SPE Middle East Oil and Gas Show and Conference* (2013), Society of Petroleum Engineers.

[50] Hill, Z.; Humphrey, M. A quantitative analysis of high performance computing with amazon's ec2 infrastructure: The death of the local cluster? In *Grid Computing, 2009 10th IEEE/ACM International Conference on* (2009), IEEE, pp. 26–33.

[51] Hochstein, L.; Basili, V. R.; Vishkin, U.; Gilbert, J. A pilot study to compare programming effort for two parallel programming models. *Journal of Systems and Software 81*, 11 (2008), 1920–1930.

[52] Islam, S.; Keung, J.; Lee, K.; Liu, A. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems 28*, 1 (2012), 155–162.

[53] Jackson, K. R.; Ramakrishnan, L.; Muriki, K.; Canon, S.; Cholia, S.; Shalf, J.; Wasserman, H. J.; Wright, N. J. Performance analysis of high performance computing applications on the amazon web services cloud. In *Second International Conference on Cloud Computing Technology and Science (CloudCom)* (2010), IEEE, pp. 159–168.

[54] Jersak, L. C.; Ferreto, T. Performance-aware Server Consolidation with Adjustable Interference Levels. In *Proceedings of the 31st Annual Symposium on Applied Computing* (2016), ACM, pp. 420–425.

[55] 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 43*, 3 (2015), 403–420.

[56] Kumar, A.; Sathasivam, C.; Periyasamy, P. Virtual Machine Placement in Cloud Computing. *Indian Journal of Science and Technology 9*, 29 (2016).

[57] Lelli, J.; Faggioli, D.; Cucinotta, T.; Lipari, G. An experimental comparison of different real-time schedulers on multicore systems. *Journal of Systems and Software 85*, 10 (2012), 2405–2416.

[58] Li, Q.; Huo, Z.; Sun, N. Optimizing mpi alltoall communication of large messages in multicore clusters. In *12th International Conference on Parallel and Distributed Computing, Applications and Technologies* (2011), IEEE, pp. 257–262.

[59] Li, Y.; Tang, X.; Cai, W. Dynamic bin packing for on-demand cloud resource allocation. *Transactions on Parallel and Distributed Systems 27*, 1 (2016), 157–170.

[60] Li, Z.; Zhang, H.; OâĂŹBrien, L.; Cai, R.; Flint, S. On evaluating commercial cloud services: A systematic review. *Journal of Systems and Software 86*, 9 (2013), 2371–2393.

[61] Lin, Q.; Qi, Z.; Wu, J.; Dong, Y.; Guan, H. Optimizing Virtual Machines using Hybrid Virtualization. *Journal of Systems and Software 85*, 11 (2012), 2593–2603.

[62] Liu, H.; Li, B.; Liu, H.; Tong, X.; Liu, Q.; Wang, X.; Liu, W. The issues of prestack reverse time migration and solutions with graphic processing unit implementation. *Geophysical Prospecting 60*, 5 (2012), 906–918.

[63] Liu, W.; Nemeth, T.; Loddoch, A.; Stefani, J.; Ergas, R.; Zhuo, L.; Volz, B.; Pell, O.; Huggett, J. Anisotropic reverse-time migration using co-processors. In *SEG Annual Meeting* (2009), Society of Exploration Geophysicists.

[64] Lourenço, H. R.; Martin, O. C.; Stützle, T. *Iterated local search*. Springer, 2003.

[65] Lu, B.; Wheeler, M. F. Iterative Coupling Reservoir Simulation on High Performance Computers. *Petroleum Science 6*, 1 (2009), 43–50.

[66] Luszczek, P. R.; Bailey, D. H.; Dongarra, J. J.; Kepner, J.; Lucas, R. F.; Rabenseifner, R.; Takahashi, D. The HPC Challenge (HPCC) Benchmark Suite. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing* (2006), Citeseer, p. 213.

[67] Ma, H.; Wang, L.; Tak, B. C.; Wang, L.; Tang, C. Auto-tuning Performance of MPI Parallel Programs Using Resource Management in Container-Based Virtual Cloud. In *9th International Conference on Cloud Computing (CLOUD)* (2016), IEEE, pp. 545–552.

[68] Madhumathi, R.; Radhakrishnan, R.; Balagopalan, A. Dynamic resource allocation in cloud using bin-packing technique. In *International Conference on Advanced Computing and Communication Systems* (2015), IEEE, pp. 1–4.

[69] Maleki, S.; Gao, Y.; Garzaran, M. J.; Wong, T.; Padua, D. A. An Evaluation of Vectorizing Compilers. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)* (2011), IEEE, pp. 372–382.

[70] Martello, S.; Toth, P. Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics 28*, 1 (1990), 59–70.

[71] Martí, R. Multi-start methods. *INTERNATIONAL SERIES IN OPERATIONS RESEARCH AND MANAGEMENT SCIENCE* (2003), 355–368.

[72] Mason, C. H.; Perreault Jr, W. D. Collinearity, power, and interpretation of multiple regression analysis. *Journal of marketing research* (1991), 268–280.

[73] Mehrotra, P.; Djomehri, J.; Heistand, S.; Hood, R.; Jin, H.; Lazanoff, A.; Saini, S.; Biswas, R. Performance evaluation of amazon elastic compute cloud for nasa high-performance computing applications. *Concurrency and Computation: Practice and Experience 28*, 4 (2016), 1041–1055.

[74] Menouer, T. Solving combinatorial problems using a parallel framework. *Journal of Parallel and Distributed Computing* (2017).

[75] Min, L.; Shengke, C. Eviews statistical analysis and applications. *Electronic industry* (2011), 5–10.

[76] Mishra, M.; Sahoo, A. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *International Conference on Cloud Computing (CLOUD)* (2011), IEEE, pp. 275–282.

[77] Mury, A. R.; Schulze, B.; Licht, F. L.; de Bona, L. C.; Ferro, M. A Concurrency Mitigation Proposal for Sharing Environments: An Affinity Approach Based on Applications Classes. In *Intelligent Cloud Computing*. Springer, 2014, pp. 26–45.

[78] Nanos, A.; Koziris, N. Xen2MX: High-performance Communication in Virtualized Environments. *Journal of Systems and Software 95* (2014), 217–230.

[79] Nasim, R.; Taheri, J.; Kassler, A. Optimizing Virtual Machine Consolidation in Virtualized Datacenters Using Resource Sensitivity. In *8th International Conference on Cloud Computing Technology and Science (cloudCom2016)* (2016), IEEE.

[80] Nassif, A. B.; Ho, D.; Capretz, L. F. Towards an Early Software Estimation using Log-linear Regression and a Multilayer Perceptron Model. *Journal of Systems and Software 86*, 1 (2013), 144–160.

[81] NETTO, M. A.; CALHEIROS, R. N.; RODRIGUES, E. R.; CUNHA, R. L.; BUYYA, R. Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges. *ACM Computing Surveys 1*, 1 (2017).

[82] Ngo, T. H. D.; La Puente, C. The steps to follow in a multiple regression analysis. In *Proceedings of the SAS Global Forum Conference* (2012), Citeseer.

[83] Otto, C.; Kempka, T. Prediction of Steam Jacket Dynamics and Water Balances in Underground Coal Gasification. *Energies 10*, 6 (2017), 739.

[84] Papagiannis, A.; Nikolopoulos, D. S. Hybrid address spaces: A methodology for implementing scalable high-level programming models on non-coherent many-core architectures. *Journal of Systems and Software 97* (2014), 47–64.

[85] Peaceman, D. W. *Fundamentals of Numerical Reservoir Simulation.* Elsevier, 2000.

[86] Pires, F. L.; Barán, B. A Virtual Machine Placement Taxonomy. In *15th International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (2015), IEEE/ACM, pp. 159–168.

[87] Pires, L. F.; Baran, B. Virtual machine placement literature review. Tech. rep., Polytechnic School, University of Asuncion, 2014.

[88] Popiolek, P. F.; Mendizabal, O. M. Monitoring and Analysis of Performance Impact in Virtualized Environments. *Journal of Applied Computing Research 2*, 2 (2013), 75–82.

[89] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2016.

[90] Rameshan, N.; Navarro, L.; Monte, E.; Vlassov, V. Stay-away, protecting sensitive applications from performance interference. In *Proceedings of the 15th International Middleware Conference* (2014), ACM, pp. 301–312.

[91] Rastogi, R.; Phadke, S. Optimal aperture width selection and parallel implementation of kirchhoff migration algorithm. In *Fourth International Conference and Exposition of the Society of Petroleum Geophysicists* (2002), Citeseer, pp. 7–9.

[92] Ray, B. R.; Chowdhury, M.; Atif, U. Is high performance computing (hpc) ready to handle big data? In *International Conference on Future Network Systems and Security* (2017), Springer, pp. 97–112.

[93] Roloff, E.; Diener, M.; Gaspary, L. P.; Navaux, P. O. Hpc application performance and cost efficiency in the cloud. In *Parallel, Distributed and Network-based Processing (PDP), 2017 25th Euromicro International Conference on* (2017), IEEE, pp. 473–477.

[94] Ryan, B. F.; Joiner, B. L. *Minitab handbook*. Duxbury Press, 2001.

[95] Sadooghi, I.; Martin, J. H.; Li, T.; Brandstatter, K.; Maheshwari, K.; de Lacerda Ruivo, T. P. P.; Garzoglio, G.; Timm, S.; Zhao, Y.; Raicu, I. Understanding the performance and potential of cloud computing for scientific applications. *IEEE Transactions on Cloud Computing 5*, 2 (2017), 358–371.

[96] Shirvani, M. H.; Ghojoghi, A. Server Consolidation Schemes in Cloud Computing Environment: A Review. *European Journal of Engineering Research and Science 1*, 3 (2016).

[97] Silva, J.; Boeres, C.; Drummond, L.; Pessoa, A. A. Memory aware load balance strategy on a parallel branch-and-bound application. *Concurrency and Computation: Practice and Experience 27*, 5 (2015), 1122–1144.

[98] Singh, T.; Vara, P. K. Smart metering the clouds. In *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE'09. 18th IEEE International Workshops on* (2009), IEEE, pp. 66–71.

[99] Souza, D. S.; Santos, H. G.; Coelho, I. M. A hybrid heuristic in gpu-cpu based on scatter search for the generalized assignment problem. *Procedia Computer Science 108* (2017), 1404–1413.

[100] Steffenel, L. A.; Martinasso, M.; Trystram, D. Assessing contention effects on mpi_alltoall communications. In *International Conference on Grid and Pervasive Computing* (2007), Springer, pp. 424–435.

[101] Stolzenberg, R. M. Multiple regression analysis. *Handbook of data analysis 165* (2004), 208.

[102] Tierney, B.; Kissel, E.; Swany, M.; Pouyoul, E. Efficient Data Transfer Protocols for Big Data. In *8th International Conference on E-Science (e-Science)* (2012), IEEE, pp. 1–9.

[103] Tomić, D.; Car, Z.; Ogrizović, D. Running HPC Applications on Many Million Cores Cloud. In *40th Jubilee International Convention on Information and Communication Technology, Electronics and Microelectronics* (2017).

[104] Tsao, S.-L.; Chen, J. J. Seprof: A high-level software energy profiling tool for an embedded processor enabling power management functions. *Journal of Systems and Software 85*, 8 (2012), 1757–1769.

[105] Tsuruoka, Y. Cloud Computing-Current Status and Future Directions. *Journal of Information Processing 24*, 2 (2016), 183–194.

[106] Vandekerckhove, J.; Matzke, D.; Wagenmakers, E. Model Comparison and the Principle of Parsimony. *The Oxford Handbook of Computational and Mathematical Psychology* (2014), 300–317.

[107] Vecchiola, C.; Pandey, S.; Buyya, R. High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on* (2009), IEEE, pp. 4–16.

[108] Wijayasiriwardhane, T.; Lai, R. Component Point: A System-level Size Measure for Component-based Software Systems. *Journal of Systems and Software 83*, 12 (2010), 2456–2470.

[109] Xavier, M. G.; Neves, M. V.; Rossi, F. D.; Ferreto, T. C.; Lange, T.; De Rose, C. A. Performance Evaluation of Container-based Virtualization for High Performance Computing Environments. In *21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (2013), IEEE, pp. 233–240.

[110] Xu, C.; Chen, X.; Dick, R. P.; Mao, Z. M. Cache Contention and Application Performance Prediction for Multi-core Systems. In *International Symposium on Performance Analysis of Systems & Software (ISPASS)* (2010), IEEE, pp. 76–86.

[111] Xu, R.; Hugues, M.; Calandra, H.; Chandrasekaran, S.; Chapman, B. Accelerating kirchhoff migration on gpu using directives. In *First Workshop on Accelerator Programming using Directives* (2014), IEEE, pp. 37–46.

[112] Yerneni, S.; Phadke, S.; Bhardwaj, D.; Chakraborty, S.; Rastogi, R. Imaging subsurface geology with seismic migration on a computing cluster. *Current Science 88*, 3 (2005), 468–474.

[113] Yokoyama, D.; Schulze, B.; Kloh, H.; Bandini, M.; Rebello, V. Affinity aware scheduling model of cluster nodes in private clouds. *Journal of Network and Computer Applications 95* (2017), 94–104.

[114] Younge, A. J.; Henschel, R.; Brown, J. T.; Von Laszewski, G.; Qiu, J.; Fox, G. C. Analysis of virtualization technologies for high performance computing environments. In *International Conference on Cloud Computing (CLOUD)* (2011), IEEE, pp. 9–16.

[115] Yu, S.; Liu, H.; Chen, Z. J.; Hsieh, B.; Shao, L., et al. Gpu-based parallel reservoir simulation for large-scale simulation problems. In *SPE Europec/EAGE Annual Conference* (2012), Society of Petroleum Engineers.