Received: 9 April 2018 | Revised: 14 November 2018 | Accepted: 21 November 2018
DOI: 10.1002/cpe.5098

WILEY

SPECIAL ISSUE PAPER

Optimizing multi-tier application performance with interference and affinity-aware placement algorithms

Uillian L. Ludwig | **Miguel G. Xavier** | **Dionatrã F. Kirchoff** | **Ian B. Cezar** | **César A. F. De Rose**

Polytechnic School, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Rio Grande do Sul, Brazil

Correspondence
Uillian L. Ludwig, Polytechnic School, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Rio Grande do Sul, Brazil.
Email: uillian.ludwig@acad.pucrs.br

Present Address
PUCRS, Polytechnic School, Building 32 Av. Ipiranga, 6681 - Partenon CEP - Porto Alegre-RS Brazil

Summary

Cloud providers are constantly seeking to become more cost effective, where a common strategy is to consolidate multiple applications in physical machines using virtualization techniques. This consolidation, however, may result in performance related problems such as resource interference. Moreover, if the workload is composed of multi-tier applications, an increasingly popular method of application development, especially for web and mobile, in which tiers need to communicate through the network, we have another possible source of performance degradation, which we refer as network affinity. In order to reduce the effects of such problems, placement techniques are used to better distribute the applications in the physical machines. Several of these placement techniques consider resource interference or network affinity in order to decide the best placement, however, none of them apply both criteria at the same time. In our previous work, we identified that a combined approach could result in better solutions for this problem and proposed a set of placement policies that explore this tradeoff. In this paper, we propose placement algorithms based on these policies and evaluate the proposed solutions for different workload scenarios using a visual simulation tool we developed called CIAPA. CIAPA introduces a performance degradation model, a cost function, and heuristics to find a placement with the minimum cost for a specific workload of multi-tier applications. In our preliminary experiments, we compared the solution generated by CIAPA with other placement strategies from related work, and have verified that, for the tested scenarios, it delivers placement decisions with better cost and, consequently, improved performance. We observed a reduction in response time of 10% when compared to interference strategies and up to 18% when considering only affinity strategies.

KEYWORDS

application placement, multi-tier applications, network affinity, performance interference

1 | INTRODUCTION

With the popularization of pay-per-use virtualized infrastructures, such as public clouds, users have a very interesting alternative to deploy their critical applications with a reduced initial investment and the possibility to easily scale in periods of increased demand.[1]

Cloud providers, however, seeking to become more cost effective, use consolidate environments to provide this service, relying on virtualization strategies, so multiple applications transparently share the same resources of a cluster of physical machines (as illustrated in Figure 1). Even though the resource sharing techniques have evolved, this may still result in overhead such as resource interference, which may severely degrade the applications' performance.

Regarding workload, multi-tier applications have become a very popular method of development of several types of applications, including web and mobile,[2] because of their improved reusability and maintainability.[3] These are applications of high importance to the society, where

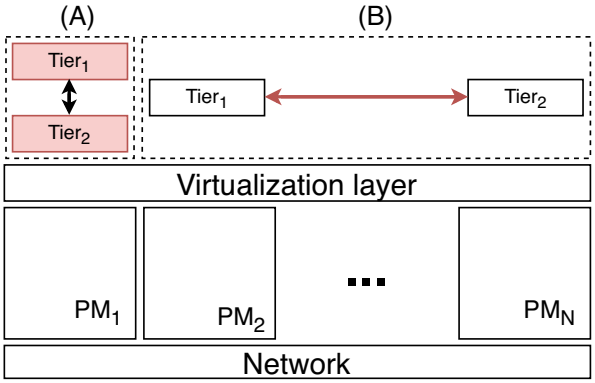

FIGURE 1 Two placements of the same multi-tier application: both tiers placed in the same physical machine and therefore generating interference (A) vs tiers placed in two distinct physical machines resulting in communication overhead (B)

several companies relying on them to operate their business. The performance of these applications is a key factor for companies' revenue because application slowdowns or downtimes lead to unsatisfied clients and, consequently, revenue loss.

In addition to the consolidation overhead, the execution of a multi-tier architecture in these infrastructures generates a great deal of pressure on the network, since the application is broken down into several specialized modules, called tiers, that have to communicate with each other to finalize the requested task.[4] Even when a conscious decision to lose some performance by restricting consolidation to improve availability/scalability is made during deployment, our model could be used to quantify this performance loss and better analyze this tradeoff.

Based on this, the main objective of this paper is to optimize the performance of multi-tier applications that are executed in consolidated environments. This optimization will be done by generating a better distribution of the applications tiers over the physical machines so that resource interference and network bottlenecks are minimized.

In order to better illustrate this problem, Figure 1 shows two placements of the same multi-tier application with two tiers. In placement *a*, the application is suffering from performance degradation due to resource interference, because both tiers are stressing the same resources of a physical machine (represented with the red background). On the other hand, in placement *b*, the application is suffering from performance degradation due to network overhead, because the tiers are placed in different physical machines and have to communicate over the network (represented with the red arrow).

The challenge here and the goal of this work is not to eliminate performance degradation but rather to find the placement that leads to the lowest performance degradation possible. We accomplish that with a strategy that repels applications that will stress the same resource from the physical node and attracts two modules to the same multi-tier application that communicate a lot to the same physical node.

In this context, we propose placement algorithms based on these policies and evaluate the proposed solutions for different workload scenarios using a visual simulation tool we developed called CIAPA (Capacity, Interference and Affinity-aware Placement Algorithms). CIAPA uses a performance degradation model, a cost function, and heuristics to find a placement with the minimum cost for a specific workload of multi-tier applications.

Our simulation results show that when these two aspects, interference and affinity, are combined in a placement strategy, which is a novel approach, resources are better utilized, and the performance of the consolidated applications tend to improve, and this is the context and the main contribution of this work.

2 | BACKGROUND

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

2.1 | Application placement

In multi-tenant environments, such as clouds, a physical machine (PM) shares its resources in order to host several related or unrelated applications. These environments are also know as consolidate environments due to the fact that they consolidate multiple applications into a single PM. Moreover, the resource sharing is done through the use of specific techniques, such as virtualization and containerization.[5] These techniques help to keep the infrastructure costs low since it enables a higher number of applications to be deployed in the same PM. However, since the PMs' resources are limited, there needs to be a definition of how many and which applications will be hosted in each PM. Therefore, application placement strategies are valuable to find the best distribution of applications per physical machines.

The main goal of placement strategies is to use the available infrastructure in the most efficient way. This efficiency, however, is seen differently by each cloud provider. As examples, some providers are concerned in reducing costs, so they aim at placing the most applications per PM. On the other hand, some providers want to minimize the service-level agreement (SLA) violations, so they aim at a placement focused in performance.[6] Furthermore, as we can see, the provider must choose between cost reduction or performance improvement, as they are conflicting goals. However, in either way, the use of smart placement strategies is important to meet the desired objective.

Ideally, placement strategies should share these following objectives[6]: Reduction of data-center traffic and distribution of this traffic evenly in the data-center network; maximization of resource utilization to effectively use every PM; minimization of the number of active PMs, which leads to a reduction of energy consumption and costs; minimization of service level agreement (SLA) violations; finally, improvement of application locality, placing applications closer to the user.

In addition, Masdari et al classify placement strategies in four groups.[6] The first group is *resource-aware*, where it takes into consideration the resource requirements of each application. In most of the strategies, CPU, memory, and network are the resources that are considered. In a few strategies, I/O and the number of active PMs are also considered. The second group is *power-aware*, which aims to reduce power consumption, leading to less cost and less CO_2 emission. The main factors considered are the CPU utilization, the PMs power usage, and the PMs states. The third group is *network-aware*. In this group, the strategies consider mainly the traffic between PMs, generating a placement with lower network overhead. The last group is *cost-aware*, which analyses the PM cost and cooling cost as its main factors.

All these strategies need to have efficient and automatic methods to find the solution that meets the desired objective. Some of them rely on the bin-packing problem, which considers application placement as the classical problem of fitting items in a bin. Since finding an optimal solution for the bin-packing problem is NP-hard, it is important to use heuristics that are able to find a good solution in a reasonable time.

While these bin-packing heuristics provide good results, there are cases where the goal is not only to pack applications into PMs, and in such cases, a more complex approach must be used. One of these approaches is the constraint programming, which is a programming paradigm where the solution must satisfy all of the problem's constraints. Moreover, other approaches rely on optimization strategies, and given an initial solution, they iterate in order to find the optimal solution. Some examples of such approaches would include integer programming, genetic algorithms, and simulated annealing.[7]

2.2 | Performance interference

With the use of resource sharing techniques, PMs host multiple applications. Even though the use of resource sharing techniques, such as virtualization and containerization, provide ways to fairly share resource between co-hosted applications, when multiple applications intensively use a resource at the same time, a problem of resource contention will happen. This problem is known as performance interference, and it may lead to severe performance degradation.[8-10]

Chi et al demonstrate the performance degradation caused by performance interference.[11] In their experiments, they have tested several combinations of applications running in the same PM. They have found that depending on the applications that are co-hosted, the performance varies considerably. Moreover, CPU intensive applications did not suffer much while being placed with other CPU intensive applications. On the other hand, disk I/O intensive applications suffered a great deal while being placed on other disk I/O applications.

Jersak and Ferreto showed in their experiments how CPU, memory, and disk I/O intensive applications are affected by other virtual machines (VMs) using the same resource intensively in the same PM.[12] They also have verified that each resource is affected differently. For example, the addition of several CPU intensive applications led to a performance degradation of 14%. On the other hand, for memory and disk I/O intensive applications, the performance degradation was as high as 90%. Therefore, it is clear that performance interference is a problem, and the performance degradation varies depending on the resource that is used the most.

In addition to VMs being affected by performance interference, Xavier et al studied how performance interference affects container-based clouds.[13] They analyzed the performance degradation suffered on disk-intensive applications caused by containers that use different resources intensively. They have tested several combinations of workloads running on the same host. While some of these combinations led to a performance degradation of 38%, they could also combine the workloads with no interference. Thus, it shows the importance of understanding each application's workload and smartly placing them in order to minimize interference.

2.3 | Multi-tier architecture

In the past, applications were commonly implemented in the client/server style, where the client was a desktop application and the server was a database running remotely. This client/server architecture is still used by simple applications, but as applications have grown more complex this basic architecture is not able to handle such complexity.[14] Therefore, this client/server architecture has adapted to what we know today as multi-tier architecture.[2]

In a multi-tier architecture, the application is broken down into several modules, called tiers. Each tier is responsible for executing a specific task, such as handling user authentication or parsing logs. Moreover, tiers may depend on other tiers to execute, and they must communicate through the network in order to answer each request.[2] For these tiers that communicate, we say that they have a network affinity relationship,

and the higher the communication rate/size is, higher the network affinity is. Moreover, this communication plays a major role in the applications' performance, and network delays may cause significant slowdown. Therefore, this complexity of network dependency of tiers puts a high pressure on having a good management of the infrastructure so that network bottlenecks can be avoided.[3]

With the rise of applications running on the cloud, including the multi-tier ones, cloud data centers have become more concerned with network utilization. An overloaded network impacts the cloud in several ways, such as increase of infrastructure costs and reduction of Quality of Service (QoS).[4] Moreover, as multi-tier applications are composed of several individual applications and as clouds host multiple applications in the same PM, resource interference may be another factor of concern. Therefore, given these problems, it is not only important for cloud providers to use techniques to reduce network traffic, placing applications close to the data they need, but it is also important to avoid placing applications that use the same resource intensively in the same PM.

3 | PLACEMENT POLICIES

In this section, we detail the first experiments that we have done in this work, and that were published at the XVIII WSCAD conference.[15] Here, we focus on the creation of a set of placement policies that would aid administrators to make placement decisions. In order to achieve that, we first characterize the interference and affinity metrics and the tools we have used to execute and measure it. Moreover, we analyzed the performance of multi-tier applications that use CPU and Disk I/O intensively and then generate the placement policies based on such analysis.

3.1 | Interference and affinity characterization

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

3.1.1 | Node-tiers: a multi-tier benchmark

As the first challenge, we had to decide which multi-tier application we would use as our target. We analyzed the characteristics of the benchmarks that are used in other research projects in the literature, and we found that none of them were suitable for our purposes. We needed a multi-tier benchmark, but most of the benchmarks used to characterize performance interference are not implemented in this architecture. Most of them execute a large task, which usually takes up to several minutes to finish and stress a specific resource, and the most popular benchmark is stress-ng.[16] However, the tasks in multi-tier applications take a short time to finish, usually less than a second. The high load comes from the high number of requests arriving at the same time, rather than the execution of a big single task. Thus, this type of benchmarks would not be useful for characterizing multi-tier applications.

Given the lack of an appropriate benchmark, in order to meet our research necessities, and, perhaps, to help other researchers, we have built an open-source multi-tier benchmark called node-tiers[†] Node-tiers was based on the stress-ng benchmark suite, and it allows us to simulate a multi-tier application with any number of tiers, where each tier executes different tasks, using different computing resources. Moreover, node-tiers is able to simulate the communication between tiers, where it is possible to set the amount of data that is being communicated on each request. Thus, node-tiers allows simulating a wide variety of applications workloads.

Node-tiers allows to simulate some types of network topologies, such as point-to-point and ring. Also, it is easily extensible, and as it being open-source, it can be expanded to support other complex structures. In our experiments, however, we were more concerned about a point-to-point architecture, with multiple tiers. Therefore, its current architecture was a good fit for our experiments.

3.1.2 | IntP: An Interference Profiler

In order to characterize the interference generated by each application tier, we used a tool called IntP. IntP is an interference profiling tool developed by Xavier as part of his PhD in Computer Science.[17] IntP monitors an application process that it expects as a parameter. It profiles the application during runtime, returning the interference the application generates on each resource subsystem. Moreover, IntP returns the interference metrics, in percentage, every second, where the higher the metric is, the more interference the application being profiled generates. IntP differs from other resource usage tools since it inspects the internal components of the operating system to infer contention-related performance interference due to bottlenecks in I/O queues, buffers, and uncore buses. From the point of view of memory consumption, an application that allocates 80% of memory would not imply that it is stressing the memory, as it could just have it allocated and not doing many

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

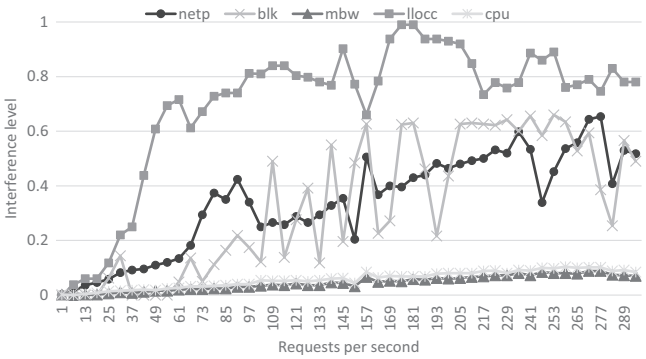

FIGURE 2 Example of an IntP output for a disk intensive application

operations. On the other hand, an application that is using only 20% could be doing a great amount of reading and writing operations to the memory; thus, it would generate a higher interference. These interference levels, though, are measured by the intP profiler.

IntP returns the interference metrics for CPU, disk, memory, network, and cache. More specifically, it returns the following metrics: **netp** - percentage of physical network interference; **nets** - percentage of network queue interference; **blk** - percentage of disk interference; **mbw** - percentage of memory bus interference; **llcmr** - percentage of cache miss; **llocc** - percentage of cache interference; lastly, **cpu** - percentage of cpu interference. Figure 2 shows the interference levels of an application while varying its workload from 0 to 300 requests per second. This application is disk intensive that has a high network affinity with another application, and it was simulated using the node-tiers benchmark. It is noticeable that the interference levels tend to increase as the workload also increases. Moreover, the disk interference has a unique behavior, which varies between every data collection. This behavior is due to writing operations being executed in an asynchronous manner.

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

3.2 | Multi-tier application performance analysis

For the performance analysis, we are using the node-tiers benchmark, considering three multi-tier applications with two tiers each, where both tiers stress the same resource. The first application was CPU-intensive, the second was disk-intensive, and the last did not use any resource intensively. Moreover, we generated an increasing workload, varying the request rate from 0 to 300 requests per second. This variation directly impacts the resource interference and network affinity levels since higher request rate leads to more resources used to answer the requests. Furthermore, we have considered two placement variations, where in the first both tiers were placed in the same PM and in the second each tier was placed in a different PM. We have also considered two variations of network affinity, where in the first, the application would have a low communication affinity, where the request size was set to 1KB. In the second variation, the application had a high communication affinity, and the request size was set to 512KB. For all experiments using node-tiers, we used the environment summarized in Table 1.

Figure 3A shows the performance of an application consisted of two CPU intensive tiers. The response time axis is shown in logarithmic scale for better visualization. It can be noticed that the execution with higher request size (512KB) had a worse performance as compared with the lower request size (1KB). This is a natural behavior since the higher the request size is, the more pressure it puts on both operating system and physical network. Additionally, while the request rate was low, the performance for all executions remained stable. However, as the request

TABLE 1 Environment architecture and characteristics

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

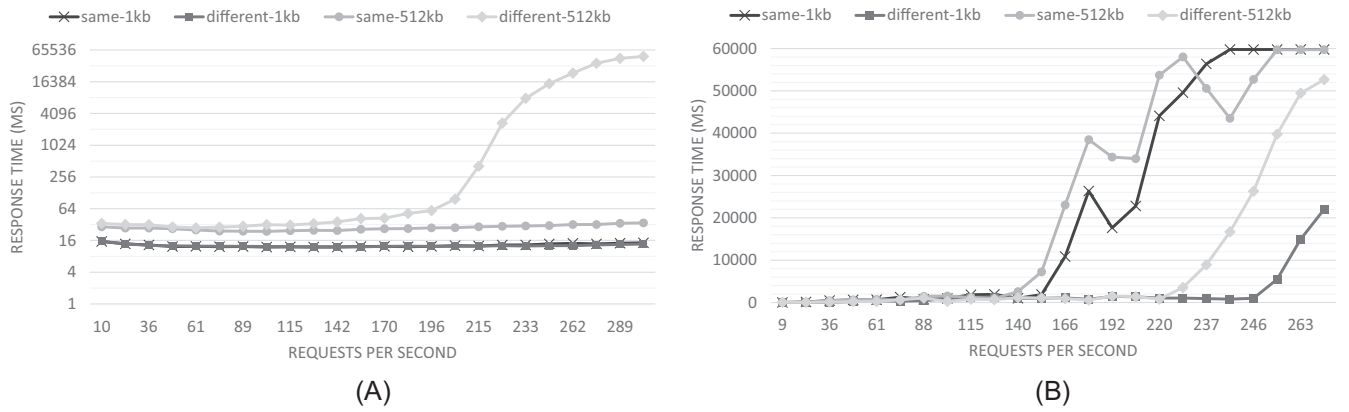

FIGURE 3 Response time of the applications while varying the workload. A, CPU-CPU; B, Disk-disk

rate increased, the execution with high network affinity running in different PMs suffered performance degradation. In this case, the network becomes flooded with many requests, and as the network bottleneck is reached, the response time increases exponentially. On the other hand, while running with same request size, but in the same PM, there is no impact on the performance. Therefore, we can conclude that, for this CPU intensive workload, network affinity is a more critical problem as compared to resource interference.

Figure 3B presents the response time of the application that had two disk I/O intensive tiers. The response time kept acceptable while the workload was low. However, as the workload increased, the application presents a different behavior from the one seen in the CPU intensive application. All four executions of this application have performance degradation, but this degradation comes earlier in the executions that run the tiers on the same PM. Furthermore, the network affinity also has an impact on the performance, and the higher the request size is, the higher the impact is. As a conclusion of this execution, disk I/O intensive applications tend to suffer more from the interference of co-hosted tiers, but there is still impact generated from different network affinity levels.

3.3 | Policies

Taking the results of the experiments into consideration, we proceeded and created a set of placement rules. As seen in the experiments, the workload has a great impact on the performance of the application. For this reason, the following rules focus on maximizing the workload that the application is able to handle without having performance degradation. Moreover, we detail these policies by describing forces that push the tiers closer or farther depending on the intensity of the interference and affinity.

- R1 Tiers that interfere the same resource should be placed in different PMs. When there are few servers available, PMs might have to host tiers that interfere the same resource. However, it should be given preference for separating the tiers that generate the most performance degradation. In this case, disk I/O tiers have a strong repulsion, while CPU intensive tiers have a weak repulsion.
- R2 Tiers that have network affinity should be placed in the same PM. When all the resources of a given PM are being used, it should be given preference to place in the same PM the tiers that have higher affinity, ie, higher request size. Moreover, the higher the affinity is, the more attraction force it generates.
- R3 Tiers that do not interfere nor have network affinity may be placed anywhere in the infrastructure. There is no force pushing the tiers.
- R4 Tiers that interfere and have network affinity should follow a sub-set of rules. These sub-rules extract the best trade-off, which should generate a placement that leads to the best quality of service (QoS) possible.
  - R4a CPU intensive tiers should be placed in the same PM. The attraction force generated by affinity is stronger than the repulsion force generated by interference.
  - R4b Disk I/O intensive tiers should be placed in separate PMs. The performance degradation that comes from interference leads to a stronger repulsion than the attraction generated by network affinity.

Even though these policies are useful for optimizing the placement, they have two main weaknesses. Firstly, it would be important to expand these policies, considering other resources, such as memory and cache. However, the policies would become too complex and hard to be understood. Secondly, these policies do not consider the levels of resource interference and network affinity. This would lead to applications to not have the optimal placement. For these reasons, in Section 4.1, we present a more advanced technique for deciding the best placement of multi-tier applications.

4 | CIAPA: CAPACITY, INTERFERENCE, AND AFFINITY-AWARE PLACEMENT ALGORITHMS

4.1 | Interference and affinity classification

We described IntP and how it provides interference levels to quantify performance interference among applications. At first glance, it seems a good idea to improve placement decisions using IntP outcomes. However, as we have seen in Section 2.2, each resource may suffer from interference in different ways. A high level of disk interference may be much more prejudicial to an application than a high level of CPU interference. For this reason, we are not going to use the interference levels by themselves, but rather the performance degradation a given interference level generates. Hence, we classified interference and affinity levels into four classes for simplification: **Absent, Low, Moderate, and High**. Even though this classification reduces the breadth of the problem, it is still an improvement to the state-of-the-art works, which most of them consider only two levels (absent and present). Each class covers different interference and affinity levels that go from 0% to 100% as presented in Table 2.

Since higher levels of interference (80%˜100%) are more difficult to achieve for any application being profiled with IntP, an equal distribution would cause most applications to be sorted at smaller intervals. That is why we distribute the levels in a more centralized and balanced way. Based on the classification, we analyzed performance interference for co-hosted applications using the node-tiers benchmark and response time as performance metric. Initially, we prepared synthetic workloads that fit an application into each of the classes described in Table 2. For the Absent class, there is no performance degradation, ie, it increases the response time in 1.0 time. For other classes, we conducted experiments running a two-tiers application, and put a load stress using the stressing tool Artillery[18] to find out the workload necessary to fit the application into the Absent, Low, Moderate, and High classes.

The interference-related performance degradation was obtained using a simulated one-tier application deployed by node-tiers. The stressing tool Artillery[18] was configured with 50 concurrent threads producing HTTP's request bursts to the application during the 40-minute runtime. We collected the average response time while the application was running in isolation. Afterwards, we inserted Low, Moderate, and High applications in the same PM, and calculated the performance degradation using the equation $perf_{class}/perf_{absent}$, where $perf_{class}$ is the average response time for each interference class, and $perf_{absent}$ is the average response time while running in isolation.

To illustrate this scenario for memory, lets take an example: We executed a memory intensive application in isolation, which resulted in $perf_{absent} = 22.8ms$. While co-hosting with a Low-intensive memory application, the runtime increased to $perf_{low} = 24.4ms$, which gives performance degradation of 1.07 times. When co-hosting with a Moderate-intensive application, the response time increases to $perf_{moderate} = 36.9ms$, resulting in a performance degradation of 1.62 times. Finally, when co-hosting with a High-intensive memory application, the response time increased to $perf_{high} = 39.7ms$, resulting in a performance degradation of 1.74 times.

In addition to interference classification, we used a similar approach to model the performance degradation related to network affinity. For this case, we used a two-tiers application that put stress only on the network subsystem. Moreover, when multiple tiers are executed in the same PM, they are running in isolation, which means that the network subsystem is not used. Hence, we placed the applications in different PMs for the Low, Moderate, and High affinity levels. The affinity levels varied according to the size of each request, where Low was 1KB, Moderate was 128KB, and High was 256KB. The performance degradation was calculated using the same method as the interference one, taking into consideration the response time of each class. Furthermore, based on this methodology, we characterized the interference and affinity performance degradation on our environment as shown in Table 3.

TABLE 2 Classification of interference and affinity levels. The higher the class, the greater the interference in performance, as well as the greater the affinity between multiple application's tiers

Class	Min	Max
Absent	0%	0%
Low	1%	20%
Moderate	21%	50%
High	51%	100%

TABLE 3 Performance degradation generated by resource interference and network affinity

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

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

4.2 | Modeling placement costs

Placement algorithms aim to put a set of applications in the smallest number of PMs to make the datacenter resource efficient. In this work, we are trying to minimize the performance degradation generated by resource interference and network overhead. This section details the notations for the placement costs considering capacity, interference and affinity. Table 4 shows the summary of notations that we used along this paper.

Capacity Constraint: the capacity cost function guarantees that the given PM P has enough capacity to host all tiers represented by the size set $S^{'}$. If P is not able to host all of them, ie, the sum of tiers sizes is greater than the capacity, it returns a high cost value, defined as ∞, which basically invalidates the given configuration; otherwise, it returns 1, which is a number that when multiplied will not modify the total cost. The capacity constraint function is defined as follows:

$$f_{cap}(S', P) = \begin{cases} \infty & \sum_S, \forall_S \in S' > P_C \\ 1 & otherwise. \end{cases} \tag{1}$$

Interference Cost: the interference cost function gives the total interference cost of running a set of tiers $T^{'}$, represented by their interference set $I^{'}$, in the same PM. The interference level for each resource is denoted as follows:

$$g(I'_{res}) = \{I | I \in I'_{res}, I > 1\} \tag{2}$$

$$f_{int'}(I'_{res}) = \begin{cases} \prod_{I \in g(I'_{res})} I & |g(I'_{res})| > 1 \\ 1 & otherwise, \end{cases} \tag{3}$$

where $res = \{CPU, memory, disk, cache\}$. The helper function g denoted in Equation (2) returns a set of values that are greater than 1, ie, which cause performance degradation. Furthermore, the function $f_{int'}$ obtains the set of values that cause performance degradation for the set of tiers I', and in the case when there is more than one tier that generates interference over the same resource, their performance degradation values are multiplied and returned as the cost; otherwise, if there is zero or one tier that interfere a given resource, it returns 1.

Finally, the total interference cost is given by the multiplication of the cost of each resource, which is calculated by using the function seen in Equation (4)

$$f_{int}(I') = f_{int'}(I'_{cpu}) * f_{int'}(I'_{memory}) * f_{int'}(I'_{disk}) * f_{int'}(I'_{cache}). \tag{4}$$

Affinity Cost: the affinity cost function returns the cost of running the set of tiers $T^{'}$, represented by their affinity set $A^{'}$, in the same PM. It iterates through each affinity entry of each tier, and by calling a helper function, seen in Equation (6), it calculates the total cost. The cost is 1

TABLE 4 Notations for the problem formulation

Symbol	Meaning
T'	A set of tiers. A tier is defined as a tuple $T = (I, A, S)$.
I	A tuple of performance degradation generated by interference. The tuple is defined as $I = (cpu, memory, disk, cache)$. These values depend on the tiers' interference levels, and they are taken from the model seen in Table 3.
A	A set of network affinity between tier T and other tiers in set T'. An element of the set A is defined as $(affinity_i, T_i)$, such that $\{affinity_i, T_i \mid i \in T'\}$. $affinity_i$ is the performance degradation given by the model seen in Table 3.
S	Size of tier T, where $S > 0$.
I'	Set of interference elements from each tier in a set T'.
A'	Set of affinity elements from each tier in a set T'.
S'	Set of size elements from each tier in a set T'.
P'	A set of PMs that will host a sub-set of tiers. A PM is defined as $P = C$.
C	Capacity of a PM P, where $C > 0$.

(no cost) when all tiers that have affinity relation are also present in the same set of tiers T', which are placed in the same PM. If a tier is not present, the returned cost is the multiplication of each performance degradation that it will have by running tiers in different PMs

$$f_{aff}(A', T') = \prod_{A \in A'} \prod_{a \in A} f_{aff'}(a, T') \tag{5}$$

$$f_{aff'}(a, T') = \begin{cases} 1 & a_T \in T' \\ a_{affinity} & otherwise \end{cases}. \tag{6}$$

Given these three cost functions, Equation (7) shows a function that returns total cost of running a set of tiers T' in a PM P, where f_{cap} works as a constraint function, measuring if the tiers fit the given PM. Then, Equation (8) shows the placement function, which tries to minimize the total cost by testing all possible combinations of tiers per PMs. Moreover, the total cost of a placement is given by the average costs of each PM. The choice for the average of costs per PM was made in order to try to keep the costs the same among all PMs. While if we would use multiplication of costs, it could lead to some PMs with cost low, while others with cost really high. Even though this is not completely avoided by using average, it is less likely to happen

$$f(T', P) = f_{int}(I') * f_{aff}(A', T') * f_{cap}(S', P) \tag{7}$$

$$p(T', P') = min(avg(f(T'_1, P_1), \ldots, f(T'_n, P_m)), avg(f(T'_2, P_1), f(T'_3, P_2)), \ldots, avg(f(T', P_1))). \tag{8}$$

The placement function should be able to return the best placement configuration; however, in order to execute it, we would have to run every combination of placement possible, which is not practical. For this reason, Section 4.3 shows the placement algorithms that we have used in order to optimize the solution.

4.3 | Placement heuristics

Given the high complexity of the placement function, heuristics are an alternative to find a solution in acceptable time. In this section, we demonstrate the use of an optimization-based strategy. For this, we use two approaches, which works as the minimization function seen in Equation (8). These heuristics start from an initial placement, and they iterate making modification in the solution in order to reduce the placement cost. The heuristics used in this approach were the stochastic hill climbing and the simulated annealing,[19] and they were used by similar works and they fit well our optimization requirements. In the following paragraphs, we are going into detail how each heuristic was implemented.

In the optimization-based heuristics, an initial state is generated, and a set of operations are executed, trying to reduce the placement cost. The initial solution is generated by calling a round robin decreasing (RRD) placement, as seen in Algorithm 1. In this strategy, the tiers are sorted decreasingly by their size, and they are placed in the PMs in a sequential order. Here, we refer to the placement distribution as a solution. Moreover, this RRD strategy is one of the most simple ones, yet it is widely used because it has virtually no computational cost.

Algorithm 1 Round robin decreasing placement algorithm

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

The first heuristic using the optimization approach implemented was the stochastic hill climbing, which its pseudo-code is shown in Algorithm 2. This heuristic expects three parameters: set of tiers, set of PMs, and number of iterations. Moreover, it starts by generating an initial solution by calling RRD algorithm. Then, for every iteration, it tries to optimize the placement cost, generating random modifications in the solution. A new solution is taken as the best if it reduces the current cost. This procedure keeps repeating for the defined number of iterations. In the end, the solution with the lowest cost is returned.

Algorithm 2 Stochastic Hill Climb heuristic

```
Data: P′, T′, iterations
Result: s_best
bestSolution = roundRobinDecreasing(P′, T′)
while iterations > 0 do
    newSolution = randomize(bestSolution)
    if newSolution.cost < bestSolution.cost then
        bestSolution = newSolution
    end
    iterations -= 1 return bestSolution
end
```

This algorithm is stochastic due to the fact that on every iteration it generates a random modification of the current solution. In a deterministic algorithm, however, it would have to explore all possible modifications from one solution, which would bring back the high complexity. Moreover, the *randomize* function, which its pseudo-code is seen in Algorithm 3, receives the current solution, and it executes a move or a swap operation. The move operation moves a random tier from one PM to another. The swap operation swaps two random tiers from different PMs. Furthermore, the move and swap operations have a probability of 50% each to be executed.

Algorithm 3 Randomize function that generates a modified solution

```
Data: solution
probability = rand(0,1)
if probability < 0.5 then
    tier_1 = getRandomTier(solution)
    tier_2 = getRandomTier(solution)
    swap(tier_1, tier_2)
else
    tier = getRandomTier(solution)
    pm = getRandomPM(solution)
    move(tier, pm)
return s
```

Hill climbing is an algorithm that returns the minimum local, and in some cases, it might not be the minimum global. This is because it only accepts a new solution if it has a lower cost; however, sometimes, it is necessary to take a worse solution in order to improve later. For this reason, we also have used a heuristic called simulated annealing, which is designed to find the global minimum. Its pseudo-code is seen in Algorithm 4, and it works in a very similar way as compared to hill climbing.

Algorithm 4 Simulated Annealing heuristic

```
Data: P′, T′, temperature, coolingRate
Result: s_best
s = roundRobinDecreasing(P′, T′)
bestSolution = s
while temperature > 1 do
    newSolution = randomize(s)
    if P(solution, newSolution, temperature) ≥ random(0, 1) then
        solution = newSolution
    end
    if solution.cost < bestSolution.cost then
        bestSolution = solution
    end
    temperature =* 1 − coolingRate
end
return bestSolution
```

Simulated annealing, instead of receiving the number of iterations as hill climbing, expects two other parameters: temperature and cooling rate. The temperature starts high, and the cooling rate determines how much it will decrease over each iteration. Furthermore, the main reason behind the use of a temperature is that, when the temperature is high, the acceptance probability function *P*, which is seen in Equation (9), will have higher changes of accepting a worse solution. This means that in the beginning, the algorithm will try several different solutions, even if

they are worse; however, as it is reaching the end of the execution, it will tend to accept only better solutions. However, like hill climbing, it had to be implemented in a stochastic manner by using the *randomize* function; therefore, even though it will optimize the placement, it may still fail at returning the minimum global at all times

$$P(s, s', T) = \begin{cases} 1 & s'_{cost} < s_{cost} \\ exp((s - s')/T) & otherwise. \end{cases} \quad (9)$$

5 | CIAPA EVALUATION

In this section, we evaluate and analyze the quality of CIAPAs placement algorithms compared to related work. First, we compare the heuristics used by CIAPA with each other based on our cost model and then the best one is used to represent CIAPA when comparing it to two well-known strategies, one based on interference and the other based on affinity. This is done with CIAPAs integrated placement simulator. At the end of this section, we present a use case analysis, where we verify the performance achieved by executing multi-tier applications under different placements in real machines. In this experiment, CIAPAs placements are compared to the placements generated by the same interference and affinity strategies used in the simulation experiments and the impact in response time is evaluated for each solution.

Before continuing, we first define four scenarios that will serve as workload in this section (for reference, detailed workload data is available online[‡]).

- **Case I**: set of 50 tiers with a mixed distribution of used resources. This configuration is used to reproduce the same level of complexity we are seeing in real industry problems and also in related work. Tiers were divided into five groups, where each group used the following resources intensively: CPU, disk I/O, memory, cache, and mixed. Moreover, three tiers of each group had network affinity with other tiers in the same group. Other two tiers also had network affinity, but with tiers from other groups. This case aims at simulating a more controlled environment, where there is an equal distribution of utilized resources;
- **Case II**: set of 50 tiers with a random distribution of used resources. Like the first case, half of these tiers have network affinity with other tiers. This case aims at simulating a more realistic workload for cloud providers, where the deployed applications have unpredictable resource interference and network affinity levels;
- **Case III**: set of two multi-tier applications with high conflict between resource interference and network affinity. The first application has two CPU moderate-intensive tiers, with a high network affinity, while the second has two disk I/O high-intensive tiers with low network affinity. Small workload to emulate a private cloud and to allow the execution in our real test bed;
- **Case IV**: set of three multi-tier application with less conflict in the same application, but high affinity levels and resource interference between tiers from different applications. Medium workload to emulate a private cloud and to allow the execution in our real test bed.

5.1 | Comparison of CIAPA optimization-based heuristics

Figure 4 shows a cost comparison among hill climbing (CIAPA-HC), simulated annealing (CIAPA-SA), and round robin decreasing (RRD), the three optimization-based heuristics used in CIAPA. Placement cost is calculated with the proposed cost model in Section 4.2 and plotted in a logarithm scale. The goal here is two find out which of the first two heuristics generated the beast placement for the defined scenarios, and show how they compare to RRD, considering that the latter is used as a baseline in related work and also as an starting point for the former. Due to the non determinism of the heuristics, the same placement configuration is not achieved in every execution. For this reason, all costs are the average of five executions of the placement algorithm, and the error bars are the standard deviation of each case.

Figure 4A shows the results for case I, while 4b shows the results for Case II. In both cases, it is noticeable that HC and SA generate much better placements than round robin, which is expected since they further improve its initial result, but have very similar costs when compared to each other, with a small advantage for SA (less than 3% in average). We believe that the lower cost achieved by HC with 10 PMs in Case I is due to the low number of executions what can be seen in the high standard deviation bar of SA for this data point.

SA cost advantage was more significant for configurations with less PMs, where consolidation was higher. This makes sense, since HC settles with a local minimum and SA does a broader exploration in the solution space, what pays of specially in this case where interference among tiers in the same PM may have a huge impact on cost.

[‡]https://uillianluiz.github.io/ciapa/tiers/[1-4]

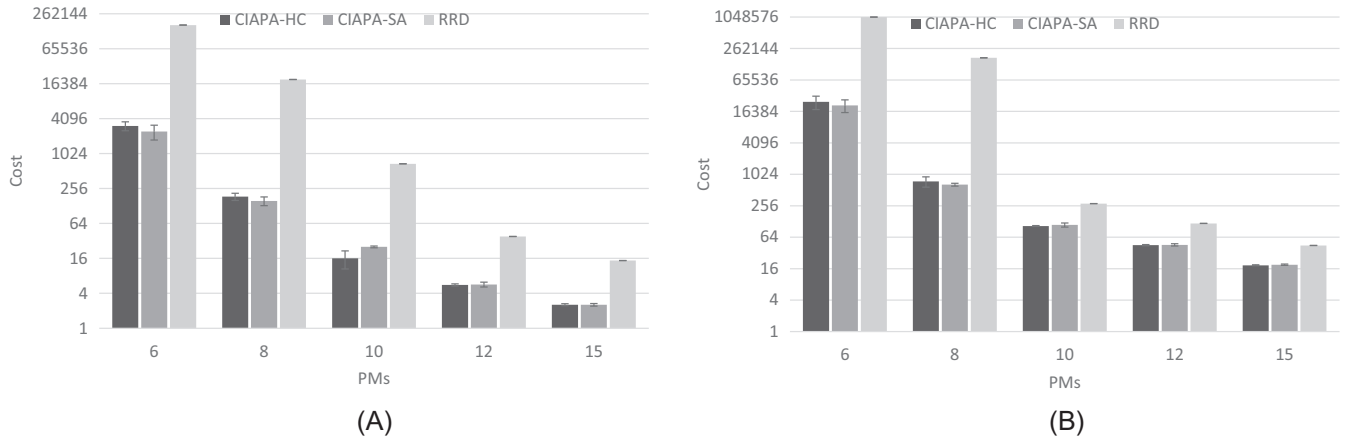

FIGURE 4 Cost comparison between CIAPAs optimization-based heuristics for Case I (A) and Case II (B)

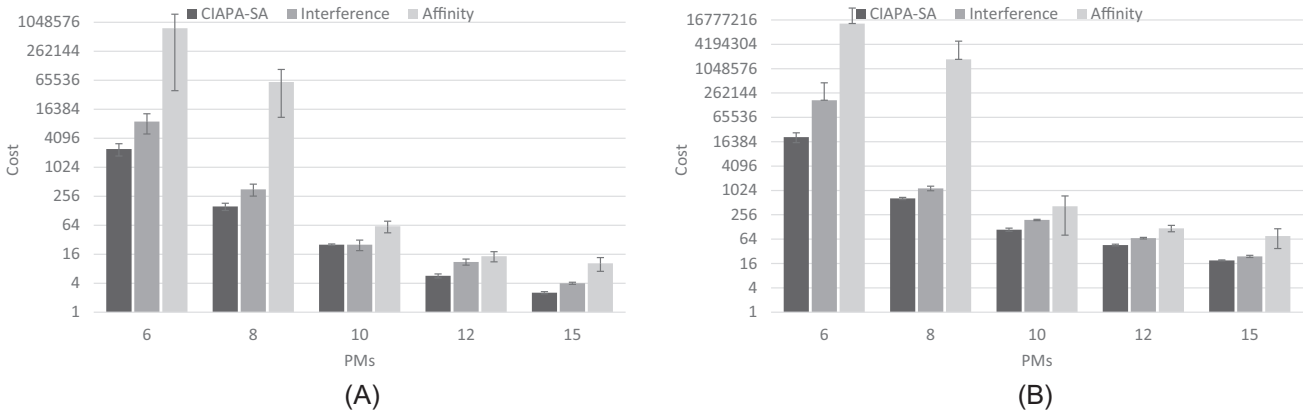

FIGURE 5 Cost comparison of CIAPA-SA against related work. A, Case I; B, Case II

5.2 | Comparison of CIAPA against related work

In this experiment, we verify how CIAPA performs when compared to related work. To represent CIAPA in this comparison we will use placements generated by the SA heuristic, due to its slightly better results as showed in Subsection 5.1. Since there is no other work that optimizes interferece and affinity at the same time like CIAPA (see Section 6), the work of Somani et al[20] was chosen to represent the interference-aware placement strategies (labeled as Interference) and the work of Su et al[21] to represent the affinity-aware (labeled as Affinity). In both cases, source code and high level algorithms were not shared by the authors so that we had to reimplement them to our experimental environment, but all characteristics of the original algorithms were maintained.

Figure 5 shows the cost comparison for placements generated by the above strategies for Cases I and II. It is noticeable that CIAPA-SA generates placement configurations with much lower cost when compared to both strategies from related work (up to 60% reduction when compared to Affinity and up to 10% to Interference). This is expected since interference has a higher impact on the cost function and Affinity strategies do not optimize for it. Results are very similar for Cases I and II demonstrating CIAPAs ability to deal with different types of workload. Also noticeable is CIAPAs significant advantage in configurations with fewer PMs. This was already seen in Section 5.1 and refers to SA being able to do a broader exploration in the solution space, what pays off specially in this case with a lot of tiers consolidated in the same PMs.

5.3 | Use case analysis

In the experiments until now, we were only concerned about the placement costs. Since these costs were calculated by our performance degradation model, they should give a good indicator of the actual impact on application's performance. However, in order to validate if the cost is actually correlated with performance, we have repeated the same experiment from Subsection 5.2 but this time with Cases III and IV in a real environment (same infrastructure used as in Section 3.2).

We executed the placement algorithms for both cases, comparing CIAPA again with the interference and affinity-aware strategies. Table 5 shows the placement distribution of tiers per PM for Case III. As expected, the interference strategy placed tiers with interference in different PMs, while the affinity strategy placed tiers with affinity in the same PM. CIAPA, on the other hand, generate a different placement, where

TABLE 5 Placement of tiers per PM generated by each strategy for Case III

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

TABLE 6 Placement of tiers per PM generated by each strategy for Case IV

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

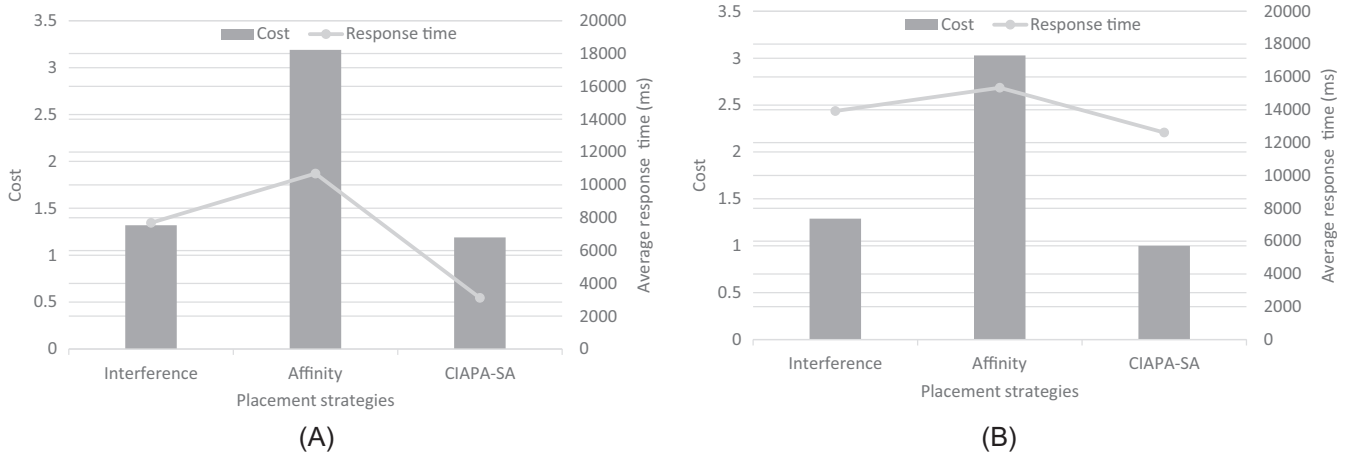

FIGURE 6 Cost and average response time comparison of CIAPA against Interference and Affinity aware placement strategies for Cases III (A) and IV (B)

three tiers were placed in one PM. This happens because CIAPA looks for the best trade-off between interference and affinity, generating the placement with the lowest cost.

In Case IV (Table 6), we have a similar behavior, where strategies that only look at one criteria generate a placement with higher cost. Considering interference-awareness, tier #3 has performance degradation because it needs to communicate with tier #4, but it was placed in a different PM. Considering affinity-awareness, tiers #2 and #4 have disk I/O interference; therefore, they have a high performance degradation for being placed in the same PM. On the other hand, CIAPA-SA is able to identify both situations and consequently generates a placement with less performance degradation.

Using these placements, we executed the actual applications in our experimental environment. Figure 6 shows the cost generated by each placement as well as the average response time achieved by the multi-tier applications for both cases. Also here, we can see that CIAPA was not only able to generate a placement with lower cost, but this cost also led to the best overall application's performance. In Case IV, a more representative workload, we can observe a reduction in response time of 10% when compared to Interference strategies, and up to 18% when considering only affinity strategies.

Our preliminary experiments with multi-tier applications have confirmed that resource interference and network affinity are characteristics that have a high impact on applications performance, what was already demonstrated by related work. Our contribution to state of the art is that the combination of these two criteria can improve results even more, and a good balance between the two is critical to this improvement.

6 | RELATED WORK

There are several research papers in the literature whose goal is to improve application performance by applying smart placement techniques. In this section, we are going to detail the ones that, like in this paper, use performance interference or network affinity metrics.

Somani et al presented VUPIC, which is a virtual machine placement scheme that takes into consideration the performance isolation and the resource utilization in order to decide the best placement configuration.[20] They classify the VMs into three different groups based on the resources they use more intensively: CPU, disk, and network. Then, when they are making placement decisions they favor to place VMs that use

different resources or same resources but with lower intensity in the same PM. With their experiments, they demonstrate that VUPIC was able to increase performance while still maintaining good resource utilization. This is a similar approach like the one used by CIAPA for the interference aspect of the work.

Caglar et al presented the harmonious Art of Living Together (hALT), which is a middleware for placement of VMs that uses machine learning.[22] hALT monitors CPU and memory utilization and the performance of virtual machines, and using this historical data, it looks for performance interference patterns that may lead to performance degradation. These patterns are used with a neural network in order to make placement decisions that will minimize interference.

Chiang and Huang presented TRACON, which is a task and resource allocation control framework.[23] This framework focuses on reducing the interference generated by data-intensive applications. They built machine learning algorithms to infer the interference levels generated by each virtual machine. This inference is used by the scheduler algorithm to generate a distribution of VMs per PM that leads to better resource utilization. In their experiments, they demonstrated that TRACON is able to improve the applications' runtime by 50%.

Xu et al presented iAware, which is a system that tries to avoid interference on VMs when a migration process happens. This is important in order to verify if a migrated processes will indeed benefit the new host machine or if it would bring performance degradation due to the interference while migrating it.[24]

Also focusing on interference, but only on network interference, Lin and Chen focused on reducing the performance degradation caused by the network I/O interference.[25] In order to reduce this problem, they have modeled a placement scheme that takes into consideration the resource demand, the QoS, and the resource interference. With their experiments, they demonstrate that the proposed placement scheme leads to a better profit and a lower number of QoS violations than other schemes that they compared their strategy with.

Yokoyama et al present an affinity aware placement scheduling strategy,[26] which, from the perspective of this work, takes only into consideration performance interference. They consider two applications affine when they not compete for the same resources, so that they could be placed in the same PM. Our definition of affinity is different, and we consider two applications affine if they communicate using the network and therefore should be considered to be placed in the same PM.

Other two works generate placement decisions based on resource interference. Firstly, Jin et al presented CCAP, which is a VM placement for HPC cloud that takes into consideration the cache contention in order to decide the best placement.[27] Secondly, Bu et al studied an interference and locality-aware scheduling of MapReduce jobs, where a job is placed closer to the data. It also takes into consideration the interference from other jobs running in the same PM.[28] In these cases, because the workload has a communication aspect like the multi-tier applications studied in our proposal, it would be interesting to also consider affinity in the placement decisions, which was not done.

In the affinity space, Ferdaus et al focus on the network affinity of tiers in order to decide the best placement of multi-tier applications.[4] Their main goal is to reduce the network overhead of data center infrastructures. Therefore, they place tiers that communicate closer together. In their experiments, they demonstrate that their placement strategy is able to reduce network costs and therefore to increase the number of applications deployed in the same infrastructure.

Another work that focuses on network affinity is presented by Su et al.[21] In their strategy, they group applications based on their communication patterns, and those applications that have a network relation are placed in the same group. Then, when deciding on which PM to place each application, they favor placing applications from the same group on the same PM so that the network overhead across PMs are reduced.

Sonnek et al present a migration strategy that reduces the communication overhead in virtualized environments.[29] They monitor the network communication between virtual machines, and they adapt the placement according to the communication behavior. In their experiments, they used MPI applications, and they were able to improve the runtime up to 42%. In addition, it enabled a reduction of up to 85% in network communication costs.

It is clear from the presented related work that resource interference and network affinity are important metrics for making optimized placement decisions, but as we can see, none of them considers the trade-off between these two metrics, like in our proposal. Especially, when considering applications that are composed by several communicating processes, like multi-tier applications, we think this is critical to obtain the best possible optimization and, consequently, the best possible application response time.

7 | CONCLUSION AND FUTURE DIRECTIONS

In this paper, we explored the problem of multi-tier application placement in consolidated environments, focusing on resource interference and network affinity. While there are several research works contributing to the same problem, as presented in Section 6, there were still opportunities for optimization considering both criteria at the same time, which is a novel approach.

Based on this, we proposed placement algorithms based on the placement policies presented in our previous work and evaluated them for different workload scenarios using a simulation tool we developed called CIAPA (Capacity, Interference and Affinity-aware Placement Algorithms). CIAPA introduced a performance degradation model, a cost function, and heuristics to find an optimized placement for a specific workload of multi-tier applications.

In our preliminary experiments, we confirmed what was already known from related work, namely that resource interference and network affinity have a high impact on application performance. However, on top of that, our simulation results showed that, when these two aspects

are combined in a placement strategy, resources are better utilized, and the performance of the consolidated applications tend to improve. We observed a reduction in response time of 10% when compared to Interference strategies and up to 18% when considering only affinity strategies.

The optimization strategies presented in this paper were focused on multi-tier applications, but their general ideas could be also applied for parallel and distributed applications that share the same model of communicating tasks if deployed in consolidate environments. The node tiers benchmark is highly configurable and could be adapted to emulate also these types of applications.

For future work, we are planning to improve our cost model and the placement heuristics, with regards to better characterize interference under different workload variations. This will probably lead to a dynamic strategy that would react to workload changes during execution time and recommend new placements.

ACKNOWLEDGMENTS

This work is sponsored by Dell for research efforts under Dell Monitoring Project. The views expressed in this paper are those of the authors and does not reflect the official policy or position of our sponsor.

ORCID

Uillian L. Ludwig https://orcid.org/0000-0002-0770-8920

REFERENCES

1. Mell Peter M, Grance T. *SP800-145 NIST Definition of Cloud Computing*. Gaithersburg, MD: National Institute of Standards and Technology; 2011. Technical Report.
2. Christoph H, Heinz S, Ralf-Detlef K. *The Distributed Architecture of Multi-Tiered Enterprise Applications*. Berlin, Germany: FREIEN UNIVERSITÄT BERLIN; 2003.
3. Huang D, He B, Miao C. A survey of adaptive resource management in multi-tier web applications. *IEEE Commun Surv Tutor*. 2004;16(3):1574-1590.
4. Ferdaus MH, Murshed M, Calheiros RN, Buyya R. An algorithm for network and data-aware placement of multi-tier applications in cloud data centers. *J Netw Comput Appl*. 2017;98(C):65-83.
5. Dua R, Raja AR, Kakadia D. Virtualization vs containerization to support PaaS. Paper presented at: 2014 IEEE International Conference on Cloud Engineering; 2014; Boston, MA.
6. Masdari M, Navabi SS, Ahmadi V. An overview of virtual machine placement schemes in cloud computing. *J Netw Comput Appl*. 2016;66:106-127.
7. Usmani Z, Singh S. A survey of virtual machine placement techniques in a cloud data center. *Procedia Comput Sci*. 2016;78:491-498.
8. Yang L, Dai Y, Zhang B. MapReduce scheduler by characterizing performance interference. *China Commun*. 2016;13(10):253-262.
9. Amri S, Hamdi H, Brahmi Z. Inter-VM interference in cloud environments: A survey. Paper presented at: IEEE/ACS 14th International Conference on Computer Systems and Applications; 2017; Hammamet, Tunisia.
10. Amannejad Y, Krishnamurthy D, Far B. Detecting performance interference in cloud-based web services. Paper presented at: 2015 IFIP/IEEE International Symposium on Integrated Network Management; 2015; Ottawa, Canada.
11. Chi R, Qian Z, Lu S. Be a good neighbour: characterizing performance interference of virtual machines under xen virtualization environments. Paper presented at: 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS); 2014; Hsinchu, Taiwan.
12. Jersak C, Ferreto T. Performance-aware server consolidation with adjustable interference levels. Paper presented at: 31st Annual ACM Symposium on Applied Computing; 2016; Pisa, Italy.
13. Xavier MG, De Oliveira IC, Rossi FD, Dos Passos RD, Matteussi KJ, De Rose CAF. A performance isolation analysis of disk-intensive workloads on container-based clouds. Paper presented at: 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing; 2015; Turku, Finland.
14. Data Abstract. Why Multi-Tier? https://docs.dataabstract.com/Introduction/WhyMultiTier/. 2017. Accessed March 5, 2017.
15. Ludwig UL, Kirchoff DF, Cezar IB, De Rose CAF. Policies for interference and affinity-aware placement of multi-tier applications in private cloud infrastructures. Paper presented at: XVIII Simpósio em Sistemas Computacionais de Alto Desempenho-WSCAD; 2017; São Paulo, Brazil.
16. King CI. Stress-ng. http://kernel.ubuntu.com/~cking/stress-ng/. 2017. Accessed November 10, 2017.
17. Xavier MG, Ludwig UL, De Rose CAF. IntP: Quantifying cross-application interference in SMP machines via resource-driven instrumentation. Paper presented at: 16th International Conference on High Performance Computing and Simulation (HPCS 2018); 2018; Orleans, France.
18. Shoreditch Ops Ltd. Artillery. https://artillery.io/. Accessed June 5, 2017.
19. Lim A, Rodrigues B, Zhang X. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *Eur J Oper Res*. 2006;174(3):1459-1478.
20. Somani G, Khandelwal P, Phatnani K. VUPIC: Virtual machine usage based placement in IaaS cloud. 2012. arXiv preprint arXiv:1212.0085.
21. Su Kui X, Lei CC, Chen W, Wang Z. Affinity and conflict-aware placement of virtual machines in heterogeneous data centers. Paper presented at: IEEE 12th International Symposium on Autonomous Decentralized Systems; 2015; Taichung, Taiwan.
22. Caglar F, Shekhar S, Gokhale A. Towards a performance interference-aware virtual machine placement strategy for supporting soft real-time applications in the cloud. Paper presented at: 3rd IEEE International Workshop on Real-Time and Distributed Computing in Emerging Applications; 2014; Rome, Italy.
23. Chiang C, Huang HH. TRACON: Interference-aware scheduling for data-intensive applications in virtualized environments. Paper presented at: International Conference for High Performance Computing, Networking, Storage and Analysis; 2011; Seatle, WA.

24. Xu F, Liu F, Liu L, Jin H, Li B, Li B. iaware: Making live migration of virtual machines interference-aware in the cloud. *IEEE Trans Comput*. 2014;63(12):3012-3025.
25. Lin JW, Chen CH. Interference-aware virtual machine placement in cloud computing systems. Paper presented at: International Conference on Computer and Information Science (ICCIS); 2012; Kuala Lumpur, Malaysia.
26. Yokoyama D, Schulze B, Kloh H, Bandini M, Rebello V. Affinity aware scheduling model of cluster nodes in private clouds. *J Netw Comput Appl*. 2017;95:94-104.
27. Jin H, Qin H, Wu S, Guo X. CCAP: A cache contention-aware virtual machine placement approach for HPC cloud. *Int J Parallel Program*. 2015;43(3):403-420.
28. Bu X, Rao J, Xu CZ. Interference and locality-aware task scheduling for MapReduce applications in virtual clusters. Paper presented at: 22nd International Symposium on High-Performance Parallel and Distributed Computing; 2013; New York, NY.
29. Sonnek J, Greensky J, Reutiman R, Chandra A. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. Paper presented at: 39th International Conference on Parallel Processing; 2010; San Diego, CA.

How to cite this article: Ludwig UL, Xavier MG, Kirchoff DF, Cezar IB, De Rose CAF. Optimizing multi-tier application performance with interference and affinity-aware placement algorithms. *Concurrency Computat Pract Exper*. 2019;31:e5098. https://doi.org/10.1002/cpe.5098