Policies for Interference and Affinity-Aware Placement of Multi-tier Applications in Private Cloud Infrastructures

Uillian L. Ludwig, Dionatrã F. Kirchoff, Ian B. Cezar, César A. F. De Rose

[1]Faculty of Informatics – Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Avenida Ipiranga, 6681, Prédio 32 - Partenon, RS, 90619-900

{uillian.ludwig, dionatra.kirchoff, ian.cezar}@acad.pucrs.br

cesar.derose@pucrs.br

Abstract. *Multi-tier is one of the most used architectures to create applications and easily deploy them to the cloud. In the literature, there are a great number of research works focusing on the placement of these applications. While some of these works take into consideration performance, most of them are concerned about reducing infrastructure costs. Besides, none of them take into consideration the interference and network affinity characteristics. Therefore, in this work, we create placement policies that aim to improve the performance of multi-tier applications by analyzing the interference and network affinity characteristics of each tier. These characteristics work as a force pushing tiers closer or farther depending on the interference and affinity levels. Moreover, by using these placement policies, we show that multi-tier applications can better utilize the infrastructure, using the same infrastructure but with an improved performance.*

1. Introduction

With the advent of cloud computing and the constant changes in software development, multi-tier applications have become a very popular method of development of every type of application [Christoph et al. 2003]. In this approach, the application is broken down into several modules, called tiers, and each module is responsible for executing a specific task. Moreover, modules may depend on the execution of other modules; thus, they may need to communicate in order to finalize the requested task. This strategy of development brings several advantages, such as the ease of reusability and maintainability. On the other hand, as there are multiple modules to be deployed, it becomes difficult to make placement decisions. This is not a major concern when deploying the application on public clouds since the customers do not have much control over the placement of their applications. However, on a private cloud data center, administrators in general, have total power to decide how to deploy each application, and it is their goal to take the most advantage of the computing resources. Therefore, they must employ placement techniques that place the multi-tier applications maximizing Quality of Service (QoS) and reducing infrastructure costs.

The use of shared environments, such as the virtualized ones, brings the advantage of using a large pool of resources to deploy several distinct applications [Chi et al. 2014]. If only one application would be placed in a physical machine (PM), this application would most likely operate with a nearly optimal performance. However, much of the resources available would be underutilized. In contrast, when placing multiple applications in the same PM, the resources are better utilized, but there is some performance degradation due to limited resources and performance interference. Consequently, it is important to find a good balance between applications per PM in order to maintain a good performance. Furthermore, when dealing with multi-tier applications, another concern rises, which is the fact that some tiers must communicate with each other in order to answer the requests. This communication is usually done by a physical network, and as the complexity of the applications increases, the pressure on the network also increases, causing the network to be a possible bottleneck on the applications' performance [Ferdaus et al. 2017]. Taking into consideration these two sources of performance degradation, we have the problem seen in Figure 1. In the placement a, we might have performance degradation due to interference, while in the placement b, we might have performance degradation due to network overhead.

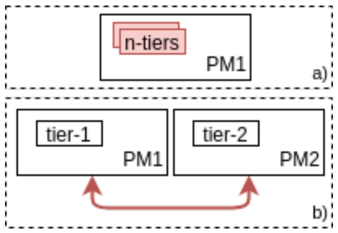

Figure 1. Conflicting placement of a multi-tier application: interference vs. affinity.

Due to this problem faced by private cloud administrators, this paper has as its main goal the creation of a set of placement policies that should lead to better application QoS and good resource utilization. The main factors that will be taken into consideration for the creation of such policies are the resource interference and network affinity of the applications. Moreover, the execution of a multi-tier benchmark with different workload and placement settings was the base the policies.

The rest of this paper is organized as follows: Section 2 goes into detail on performance interference and characteristics of multi-tier applications; Section 3 describes the placement problem and how related works aim to solve it; Section 4 shows the analysis of the multi-tier benchmark execution; Section 5 lists the placement policies; finally, Section 6 concludes the paper and describes the future work.

2. Background

2.1. Performance Interference

Shared environments, such as clouds, clusters, and data centers, are examples of multi-tenant architectures that share their resources to host several related or unrelated applications. These applications may have different purposes, such as the use for scientific research or for running the back-end of software systems. Moreover, each application uses a slice of the resource of the physical machine it is running on. The resource sharing technique varies according to each environment, but the applications may be running as processes, containers or virtual machines (VMs). Furthermore, each resource sharing technique provides different levels of resource isolation. For instance, VMs have a lower interference level between co-hosted VMs, but they have a higher overhead, while containers and processes have a higher interference level with lower overhead [Felter et al. 2015]. Therefore, it is important to understand the characteristics of the applications when deciding which resource sharing technique is most adequate.

Even though virtualization provides a good level of resource isolation, there is still some degree of resource interference between VMs running on a same physical machine (PM). This interference is due to the fact that each VM uses a slice of the resource available, such as storage, CPU, and memory. However, there are some resources that cannot be sliced, such as disk I/O, network I/O and shared cache [Ibrahim et al. 2011]. Therefore, applications running in the same PM will contend these resources, which leads to an environment that is not fully isolated, and applications may suffer performance degradation [Lin and Chen 2012].

There are several research studies that demonstrate the performance degradation that comes from the use of these multi-tenant architectures. [Chi et al. 2014] tested several workload combinations running in the same PM. The authors showed how depending on the applications that are co-hosted, the performance varies considerably. To illustrate, the application called handbrake, which is CPU intensive, does not suffer much performance degradation when co-hosted with other CPU intensive workloads. On the other hand, the application called dd, which is disk I/O intensive, suffers considerably when co-hosted with other disk I/O workloads. In addition, [Jersak and Ferreto 2016] showed that when running multiple applications that use the same resource intensively on the same host, the performance degrades considerably. They have also found a similar behavior as the previous research, where CPU had a maximum performance degradation of 14%, while for memory and disk I/O this number was as high as 90%. Finally, [Xavier et al. 2015] studied the impact of performance interference on disk intensive applications running on container-based clouds. Their experiments showed that while some combination of workloads running on the same host led to a performance degradation of 38%, they could combine workloads with no performance degradation. Therefore, these studies show how the placement of the applications can impact in the performance of the applications.

2.2. Multi-tier Application Characterization

In these multi-tenant environments, several types of workloads are executed. These workloads include high-performance systems, big data processing, multi-tier applications and so on. The performance requirements of these applications vary, but it is always desirable to ensure that the maximum performance is achieved from the resources available. In this paper, we focus specifically on multi-tier applications, which are characterized by having multiple modules that may communicate in order to execute tasks. The multi-tier architecture is widely used nowadays, but it is not a new concept. The first use of this architecture dates back to the 90s, where client/server applications were commonly used [Data Abstract 2017]. However, this architecture has suffered many changes to adequate to technology evolution and security concerns.

Web and mobile applications heavily rely on the multi-tier architecture as the main method of development [Christoph et al. 2003, Huang et al. 2004]. An important reason is the fact that the multi-tier architecture brings better maintainability and reusability [Huang et al. 2004]. Therefore, tiers may be used to provide services to both web and mobile systems due to its good reusability, and it is easier to develop and maintain since each tier will be responsible for a specific task. Moreover, a good example of a multi-tier application is the online latex editor ShareLaTeX [ShareLaTeX 2017]. This application contains several tiers that are responsible to execute specific tasks. For example, the web tier is responsible for building the web interface, the real-time tier is responsible for syncing the document changes in real time, the latex compiler tier generates the PDF files, and the spelling checking tier verifies for writing mistakes. Each tier in this application has different resource requirements, and some need higher performance guarantees than others. Thus, it is important to the service provider to ensure that the performance needed for each tier is achieved.

As the applications grow more complex, the number of tiers grow linearly; therefore, it becomes vital to have a good management of the infrastructure [Huang et al. 2004]. This management becomes harder because there are many factors that impact the workload of different applications tiers. The periodicity of the workload is one of these factors since some applications have different number of users during different periods of time such as hours and days of the week. Moreover, the number of users may impact the tiers differently. While for one tier, having a varied number of requests per second will have no impact on the performance, for another tier it might have a high impact. Furthermore, tiers need to communicate through the network in order to finish the user's requests. Thus, it is important to have a good understanding of the characteristics of each tier, making a placement that takes these characteristics into consideration.

3. Problem Statement and State-of-the-Art

Improving the performance of applications by making better placement decisions is widely studied in the literature. Most of these existing strategies use factors such as resource utilization, number of active PMs, and data-center traffic in order to decide the best placement configuration [Masdari et al. 2016]. Moreover, some works are concerned about the interference that is generated by co-hosted applications, while others are concerned about the network affinity of them. However, there is no strategy that takes into consideration both interference and affinity characteristics. Furthermore, in the following paragraphs, we describe the most relevant research studies related to both interference and affinity placement strategies.

A performance-aware placement of virtual machines was created, taking into consideration the interference levels of VMs running in the same PM [Jersak and Ferreto 2016]. The authors have built an interference model that extracts the interference levels of CPU, RAM, and disk I/O intensive applications. Based on this model, they make placement decisions, trying to minimize the interference while keeping the quantity of PMs necessary low. In addition, with the same purpose, VUPIC, which is a placement scheme that takes into consideration the performance isolation and resource utilization in order to decide the best placement, was developed [Somani et al. 2012]. It classifies the VMs in groups of interference, and those VMs which interferes the same resources intensively are placed in different PMs.

Focusing specifically on network interference, a placement scheme was implemented to reduce the performance degradation [Lin and Chen 2012]. Since network I/O is a resource that cannot be sliced, whenever multiple VMs are using it, they will contend for the same resource, leading to performance loss. Thus, they favor placing VMs that stress the network in different PMs. Moreover, using a different approach, a network affinity aware placement was built [Su et al. 2015]. In this approach, the authors group VMs that have network affinity, and try to place them in the same PM; thus, reducing the network overhead across physical networks. Besides, [Ferdaus et al. 2017] considered the network affinity, and created a placement of multi-tier applications in the cloud. They could reduce network costs, as well as keeping the server well utilized.

It is clear the importance of both resource interference and network affinity strategies. However, none of the placement strategies consider both characteristics at the same time. The main reason for this is due to the fact that these are opposed characteristics, and it is not simple to find the best trade-off. For this reason, this paper focuses on reducing the resource interference while considering the network affinity. Moreover, it should keep the quality of service (QoS) high, while maintaining the number of necessary PMs low.

4. Performance Analysis

As a first step to perform the performance analysis, 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 with this architecture. Most of them execute a large task, which usually takes up to several minutes to finish and stress a specific resource. However, the tasks in multi-tier applications take a short time to finish, usually less than a second. The high load comes from the high number of requests arriving at the same time, rather than the execution of a big task. Thus, this type of benchmarks would not be useful for characterizing the multi-tier applications. Furthermore, there is a widely used multi-tier benchmark called RUBiS, which simulates an e-commerce application [OW2 Consortium 2001]. RUBiS, however, is not very flexible when it comes to customization of resource utilization, and its use is more adequate for use cases rather than to perform deep analysis.

Given the lack of an appropriate benchmark, we decided to build a multi-tier benchmark that would allow us to do fine grained customization of both resource and network utilization. We built a multi-tier benchmark that allows to use up to n tiers, where each tier might be CPU or disk intensive or non-intensive. Moreover, it is flexible and more types of tiers, such as memory or cache intensive, may be added. Besides the ability to modify the characteristic of each tier, the benchmark allows modifying the request size for each arriving request, which will directly affect the network performance. Thus, this benchmark allows us to easily test different combinations of workloads. The architecture of the benchmark is shown in Figure 2, and the source code is found in a Github repository[1].

In order to generate load in the application, we used a tool called Artillery [Shoreditch Ops Ltd 2017]. It allows us to create virtual users, and each user will execute HTTP requests in the server. Since each request by itself does not take a long time to finish, the best way to impact the resource utilization and the performance is by varying

[1]`https://github.com/uillianluiz/node-tiers`

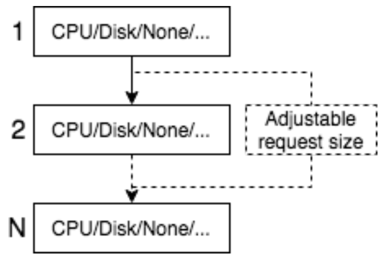

Figure 2. Multi-tier architecture of the implemented benchmark with N tiers.

the quantity of arriving requests. Thus, we configured Artillery to vary the request demand from 1 to 300 requests per second. Moreover, at the end of the processing, Artillery provides a detailed report, listing response times, requests per second processed, and so on.

The execution environment consisted of two physical machines with: 2x Intel(R) Xeon(R) CPU X6550, 64GB RAM, 128GB of storage and a gigabit ethernet connection between both of them. Each tier was set to use the maximum of 4 CPU and unlimited RAM. Moreover, we set two request size configurations for the communication of each tier: 1KB and 512KB.

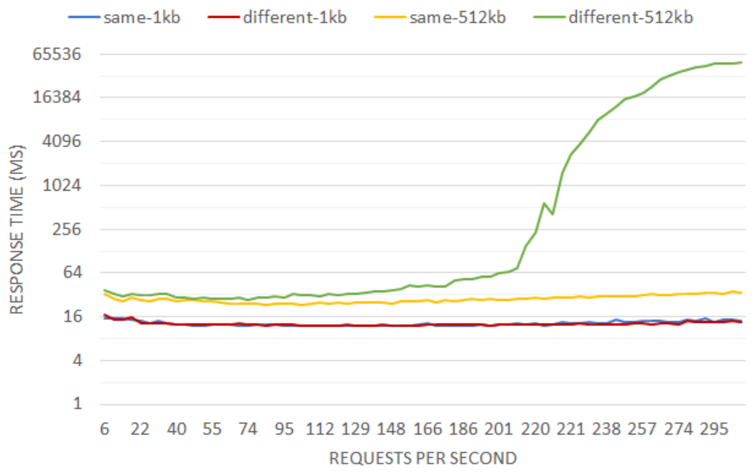

Figure 3. Response time of application CPU-CPU while varying the workload. The lines are in log scale.

Figure 3 shows the performance of an application consisted of two CPU intensive tiers. The response time axis is shown in logarithmic scale for better visualization. It can be noticed that the execution with higher request size (512KB) had a worse performance as compared with the lower request size (1KB). This is a natural behavior since the higher the request size is, the more pressure it puts on both operating system and network. Additionally, while the request rate was low, the performance for all executions remained stable. However, as the request rate increased, the execution with high request size running in different hosts suffers 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 same hosts, there is no impact on the performance. Therefore, we can conclude that the network affinity is the most impacting factor when dealing with CPU intensive tiers.

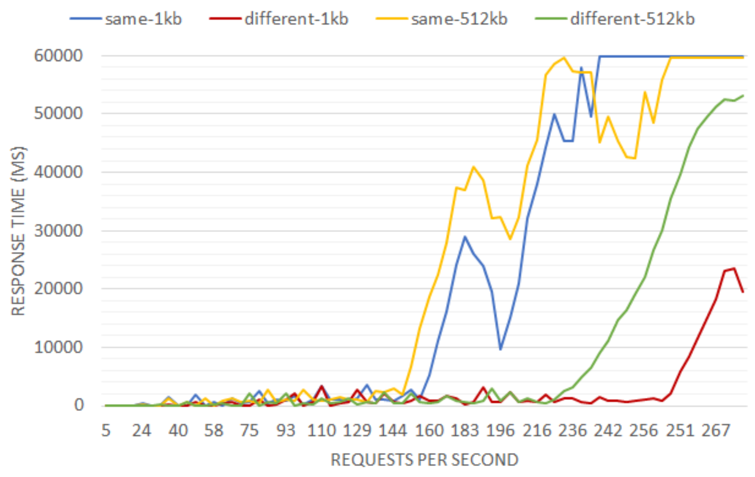

Figure 4. Response time of application Disk-Disk while varying the workload.

Figure 4 presents the response time of the application that had two disk I/O intensive tiers. The response time kept acceptable while the workload was low. However, as the workload increased, the application presents a different behavior from the one seen in the CPU intensive application. All four executions of this application have performance degradation, but this degradation comes earlier in the executions that run the tiers on the same host. 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 the network affinity levels.

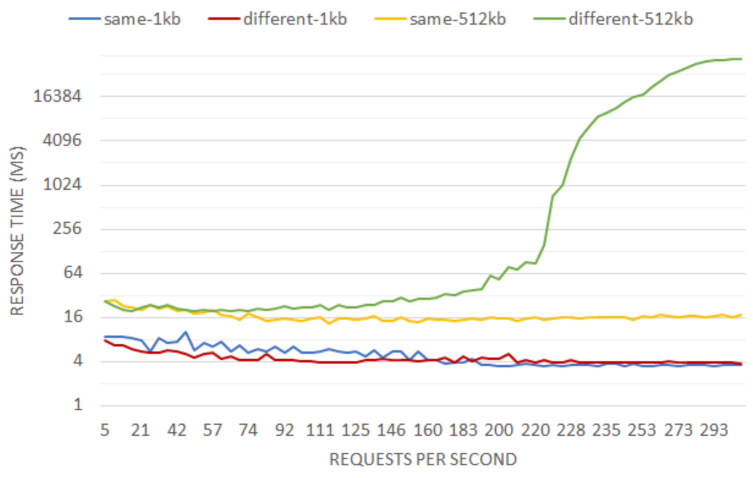

Figure 5. Response time of application Non-Intensive-Non-Intensive while varying the workload. The lines are in log scale.

Figure 5, which contains the execution of an application with two non-intensive tiers, shows a similar behavior to the one seen in the CPU intensive application. This is due to the fact that both applications are not highly affected by interference, and the characteristic that generates the most impact is the network affinity.

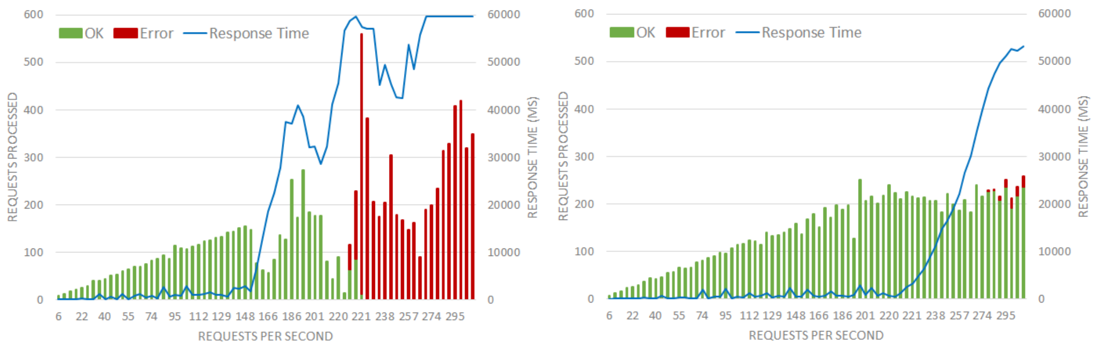

Figure 6. Impact of increasing the request rate in the response time and request status of the execution of the DISK-DISK application with 512KB of request size.

Going further into the analysis of these executions, Figure 6 shows how the requests' status was affected by the increase in the request rate. It shows the impact of the execution of the disk intensive application with 512kb of bandwidth. The chart on the left shows the results of the execution in the same host. It can be noticed that while the request rate was low, the response time was constant and the number of requests with OK status was growing linearly. However, as the request rate surpassed 150, the response time started to grow rapidly, the requests started taking longer to finish, and, finally, all of them were failing. In contrast, the chart on the right shows the execution of the same benchmark configuration but running in different hosts. As we saw in the previous analysis, the execution on different hosts had a better performance. Here, we can see that the effect on the request status matched the response time, and requests only failed when the request rate was above 270.

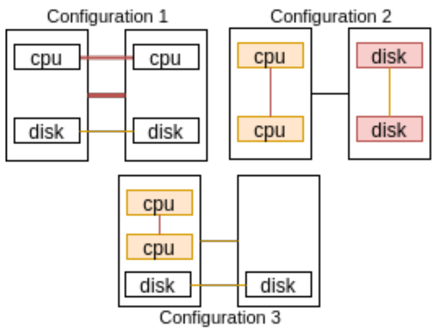

Figure 7. Different configurations of execution of two multi-tier applications.

In addition to analyzing the placement of one multi-tier application, Figure 7 shows three different placement configurations of two multi-tier applications. One application has two CPU intensive tiers, using 512KB of request size. The other one has two disk I/O intensive tiers, using 1KB of request size. For this experiment, we setup three configurations, where each one favors different characteristics. The configuration 1 favors the reduction of the interference by placing the tiers that stress the same resource in different PMs. The configuration 2 favors the network affinity by placing the tiers that communicate in the same machine. Finally, the configuration 3 tries to find the best compromise between interference and network affinity by placing the application with high affinity in the same PM and the application with high interference in different ones.

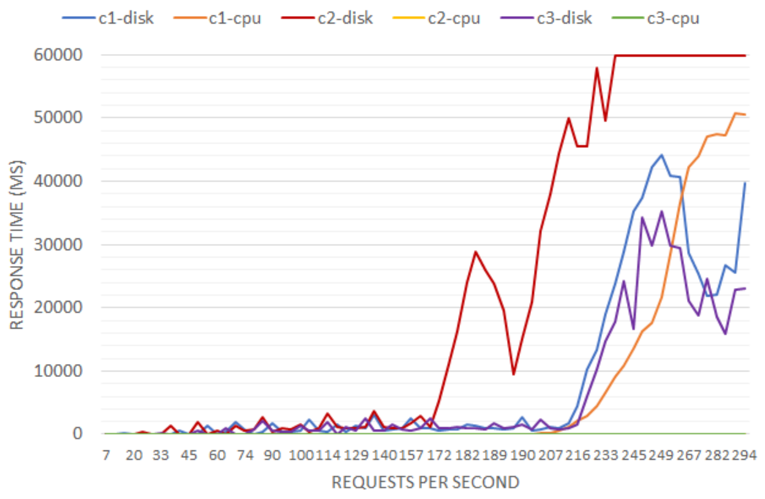

Figure 8. Response time of three combinations of placement of two multi-tier applications in two physical hosts.

The response time of the execution of the three configurations is shown in Figure 8. As seen in the previous experiments, the response time for lower request rate did not vary in the different configurations of placement. The configuration 1, which favors the interference, had a poor performance for both applications. In this case, since the CPU intensive application has a high network affinity, and the tiers were placed in different hosts, the performance is degraded. Moreover, this placement, in theory, should contribute to the performance of the disk I/O intensive application. However, it can be noticed that it also degrades. The reason behind this fact is a network interference generated from the CPU intensive application. Therefore, since the tiers of the CPU intensive application were placed in different PMs generate performance degradation on both applications placed in the infrastructure.

In addition, in the configuration 2, which favors the network affinity, the CPU intensive had no performance degradation since its tiers were placed in the same PM and there was no network overhead. On the other hand, the disk I/O intensive application had a high performance degradation due to interference, and it presented the worst performance of all executions. Finally, in the configuration 3, which tries to find the best trade-off between affinity and interference, the best performance was achieved. Even though in this configuration the quantity of tiers per host was not well-balanced, it achieved the best overall performance. The disk I/O intensive application still had performance degradation, but it was the lowest among all configurations. Therefore, these experiments showed how a better performance can be achieved by taking into consideration both interference and network affinity characteristics.

5. Placement Policies

Taking the results of the experiments into consideration, we may proceed and create 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.

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

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

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

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

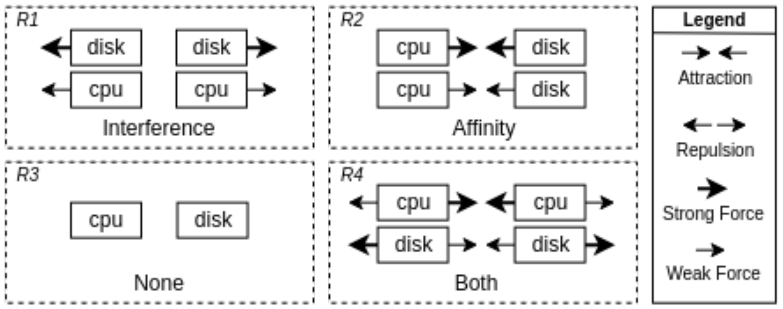

Figure 9. Illustration of the placement policies based on the forces pushing them. While interference repel tiers to different PMs, affinity attracts them to the same one.

Figure 9 shows an illustration of each rule. It demonstrate the forces that affinity and interference put in the tiers, attracting them to the same PM (affinity) or repelling them to different PMs (interference). The line width represents the strength of the force pushing them. Moreover, these rules would be enough if the infrastructure was able to scale out and up without any restrictions. However, such infrastructure does not exist, so it is important to think of how to deal whenever the resources bottleneck is reached. In this case, a way to deal with this problem is to use a business rule that gives the priority of each tier. For example, a slowdown in a tier that handles logs is less detrimental than a slowdown in a tier that handles users authentication. Therefore, tiers with less priority may be placed in the same PM even with interference, or in different PMs even with network affinity since this placement will not directly affect the business. However, tiers with high priority must follow the placement rules for achieving better QoS.

6. Conclusion and Future Work

The multi-tier architecture is undeniable one of most important methods of development of applications from all computing segments. Its wide utilization brings some important concerns related to how to achieve better performance. Besides optimizing the applications' code itself, the best way to achieve such performance is by better utilizing the infrastructure resources, which may be achieved by using smart placement techniques. Through the experiments, it was demonstrated that different placements have a great impact on the applications' performance, and that the characteristics of resource interference and network affinity should be carefully analyzed in order to decide the best placement.

By observing the difference in performance while executing the multi-tier benchmark in different placement settings, we were able to create a set of placement policies. These policies are important on determining the best placement configuration of multi-tier applications, leading to a better performance while keeping the same infrastructure. Besides, the multi-tier benchmark has a good potential, and it can be used with many purposes, such as to characterize the environment and to test placement settings.

During the experiments, we only considered CPU and disk I/O, but as future work, we plan to expand the multi-tier benchmark adding memory and cache intensive tiers. Therefore, we may be able to make decisions based on a broad type of resource intensive applications. Moreover, in addition to the placement policies, we plan to create a placement model that would receive the interference and affinity levels, and return the best placement setting for the tiers. This model will have as its base the results of this work and the experiments with the other resource intensive tiers that we will implement. After that, the model will be used to build a placement algorithm, which can be used to automate the placement process.

7. Acknowledgments

This work is sponsored by Dell Technologies Inc. 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.

References

Chi, R., Qian, Z., and Lu, S. (2014). Be a Good Neighbour: Characterizing Performance Interference of Virtual Machines Under Xen Virtualization Environments.

Christoph, H., Heinz, S., and Ralf-Detlef, K. (2003). The Distributed Architecture of Multi- Tiered Enterprise Applications. In *A Middleware Architecture for Transactional, Object-Oriented Applications*, chapter 3, pages 15–40. FREIEN UNIVERSITÄT BERLIN, Berlin.

Data Abstract (2017). Why multi-tier? https://docs.dataabstract.com/Introduction/WhyMultiTier/. Accessed: 2017-05-24.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, pages 171–172. IEEE.

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

Huang, D., He, B., and Miao, C. (2004). A Survey of Adaptive Resource Management in Multi-Tier Web Applications. *IEEE Communications Surveys & Tutorials*, 16(3):1574–1590.

Ibrahim, S., He, B., and Jin, H. (2011). Towards pay-as-you-consume cloud computing. *Proceedings - 2011 IEEE International Conference on Services Computing, SCC 2011*, pages 370–377.

Jersak, L. C. and Ferreto, T. (2016). Performance-aware server consolidation with adjustable interference levels. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 420–425. ACM.

Lin, J.-W. and Chen, C.-H. (2012). Interference-aware virtual machine placement in cloud computing systems. In *Computer & Information Science (ICCIS), 2012 International Conference on*, volume 2, pages 598–603. IEEE.

Masdari, M., Nabavi, S. S., and Ahmadi, V. (2016). An overview of virtual machine placement schemes in cloud computing. *Journal of Network and Computer Applications*, 66:106–127.

OW2 Consortium (2001). RUBiS. http://rubis.ow2.org/. Accessed: 2017-02-24.

ShareLaTeX (2017). Sharelatex. https://github.com/sharelatex/sharelatex. Accessed: 2017-07-24.

Shoreditch Ops Ltd (2017). Artillery. https://artillery.io/. Accessed: 2017-06-05.

Somani, G., Khandelwal, P., and Phatnani, K. (2012). Vupic: Virtual machine usage based placement in iaas cloud. *arXiv preprint arXiv:1212.0085.*

Su, K., Xu, L., Chen, C., Chen, W., and Wang, Z. (2015). Affinity and conflict-aware placement of virtual machines in heterogeneous data centers. In *Autonomous Decentralized Systems (ISADS), 2015 IEEE Twelfth International Symposium on*, pages 289–294. IEEE.

Xavier, M. G., De Oliveira, I. C., Rossi, F. D., Dos Passos, R. D., Matteussi, K. J., and De Rose, C. A. F. (2015). A performance isolation analysis of disk-intensive workloads on container-based clouds. *Proceedings - 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015*, (February):253–260.