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

PAULO SILAS SEVERO DE SOUZA

A HEURISTIC ALGORITHM FOR MINIMIZING SERVER MAINTENANCE TIME AND VULNERABILITY SURFACE ON DATA CENTERS

Porto Alegre

2020

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

A HEURISTIC ALGORITHM FOR MINIMIZING SERVER MAINTENANCE TIME AND VULNERABILITY SURFACE ON DATA CENTERS

PAULO SILAS SEVERO DE SOUZA

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

Advisor: Prof. Dr. Tiago Coelho Ferreto

Porto Alegre
2020

Ficha Catalográfica

S729h Souza, Paulo Silas Severo de

A heuristic algorithm for minimizing server maintenance time and vulnerability surface on data centers / Paulo Silas Severo de Souza . – 2020.
46 f.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Tiago Coelho Ferreto.

1. Cloud Computing. 2. Server Maintenance. 3. Virtualization. 4. Security. I. Ferreto, Tiago Coelho. II. Título.

Paulo Silas Severo de Souza

A Heuristic Algorithm for Minimizing Server Maintenance Time and Vulnerability Surface on Data Centers

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

Sanctioned on ________ _______, 20________.

COMMITTEE MEMBERS:

Prof. Dr. Lucas Mello Schnorr (PPGC/UFRGS)

Prof. Dr. César Augusto Fonticielha De Rose (PPGCC/PUCRS)

Prof. Dr. Tiago Coelho Ferreto (PPGCC/PUCRS - Advisor)

UM ALGORITMO HEURÍSTICO PARA MINIMIZAR TEMPO DE MANUTENÇÃO DE SERVIDORES E SUPERFÍCIE DE VULNERABILIDADE EM DATA CENTERS

RESUMO

Redução de custos e escalabilidade impulsionaram a adoção da computação em nuvem por diferentes organizações. Para manter as funcionalidades prometidas, operadores realizam diversas atividades de manutenção, que vão desde a remoção do acúmulo de poeira a correções de segurança nos servidores contra vulnerabilidades. Correções de segurança geralmente exigem que a atualização dos equipamentos afetados seja realizada o mais rápido possível, pois cada instante de espera pode indicar uma oportunidade para invasão. Soluções atuais empregam diferentes abordagens para minimizar a duração da manutenção. No entanto, tais estudos não consideram o tempo em que os servidores ficam expostos a ataques. Neste estudo, argumenta-se que apenas minimizar o tempo de manutenção não garante necessariamente a eficiência de estratégias de manutenção em cenários críticos de segurança, onde proteger os servidores o mais rápido possível é a prioridade. Portanto, propõe-se uma nova métrica chamada Superfície de Vulnerabilidade, que permite quantificar a eficiência de estratégias de manutenção em cenários críticos de segurança. Também é apresentada uma heurística que realiza decisões de manutenção para minimizar a Superfície de Vulnerabilidade e o tempo de manutenção. Foram realizados diversos experimentos e os resultados mostraram a eficácia da solução proposta em reduzir a Superfície da Vulnerabilidade, tempo de manutenção e número de migrações.

Palavras-Chave: Computação em Nuvem, Manutenção de Servidores, Virtualização, Segurança.

A HEURISTIC ALGORITHM FOR MINIMIZING SERVER MAINTENANCE TIME AND VULNERABILITY SURFACE ON DATA CENTERS

ABSTRACT

Cost reduction and enhanced scalability boosted the adoption of cloud computing by multi-sized companies. To maintain the promised features, cloud operators perform several maintenance activities that range from removing dust accumulation to applying security patches on servers against vulnerabilities. The latter usually requires server update as soon as possible, as each instant servers need to wait for an update can indicate an opportunity for attackers to breach customers' applications. Current solutions employ different approaches to minimize maintenance duration. However, they neglect the amount of time servers stay exposed to attacks. In this study, we first argue that only reducing maintenance time does not necessarily guarantee the efficiency of maintenance strategies on critical security patching scenarios, wherein safeguarding servers as soon as possible is the priority. Therefore, we propose a new metric called Vulnerability Surface, which aids operators in assessing the efficiency of maintenance strategies on critical security patching scenarios. Then, we present a heuristic algorithm that performs maintenance decisions to minimize the amount of time cloud servers remain exposed to attacks while reducing the amount of time necessary for performing server maintenance. We conducted a set of experiments against well-known strategies, and the results showed that the proposed solution achieves superior results regarding vulnerability surface, maintenance time, and the number of migrations per virtual machine.

Keywords: Cloud Computing, Server Maintenance, Virtualization, Security.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

1. INTRODUCTION

Cloud Computing enables the access to computing resources from anywhere through the Internet without requiring any commitment of customers with physical infrastructure since supplies are delivered as services [2]. Cloud's delivery model also provides features such as elasticity and scalability, which minimize resource wastage and allow customers to pay only for what they use [13]. As a result, individuals and organizations are moving applications to the cloud. To ensure cloud environments deliver the promised features, operators must wisely perform maintenance activities to safeguard the data center's health while providing the accorded quality of service [34].

Many events may produce failures inside data centers, affecting the quality of service (QoS) delivered to end-users. Aside from unintentional failures caused by issues like capacity overload due or code bugs, server attacks are becoming highly frequent [17] [14]. Examples of attacks against cloud servers are many. One could program bots to perform Distributed Denial of Service (DDoS) attacks [33]. Another example could be malicious software designed to get unauthorized access to machines to steal data or even use an internal server as a start point for attacks on other servers, and so on [4] [25].

Due to the gravity of recent attacks against cloud servers, even popular cloud providers such as Microsoft Azure[1], Amazon Web Services[2], and Google Cloud Platform[3] have reported that in some situations their operators had to focus all their attention on performing maintenance on their data centers to safeguard servers against attackers. In 2018, after the public disclosure of attacks against CPU vulnerabilities [17] [14], In its report, Microsoft Azure informed that maintenance activities had to be accelerated in such a way that some availability zones suffered from downtime during the update.

Quickly responding to attacks is outmost to preserve data centers in operable condition, but also for retaining reliability and reputation. In such a scenario, operators must define a range of parameters that will determine *"when"* the maintenance will be performed, *"which"* servers will be updated at the time, and *"how"* each one of them will be prepared to undergo maintenance.

1.1 Contribution

Previous investigations provide solutions to optimize server maintenance by means of reducing virtual machine migrations and minimizing the maintenance time [34] [31] [35] [27].

[1]https://azure.microsoft.com/en-us/blog/securing-azure-customers-from-cpu-vulnerability/.

[2]https://aws.amazon.com/pt/security/security-bulletins/AWS-2018-013/.

[3]https://blog.google/topics/google-cloud/what-google-cloud-g-suite-and-chrome-customers-need-know-about-industry-wide-cpu-vulnerability/.

However, they do not assess the efficiency of their proposals on safeguarding cloud servers as soon as possible. Therefore, in this study, we intend to address this gap by supporting cloud operators' decision-making on critical security servers maintenance.

Throughout this study, we show that reducing the maintenance time does not necessarily guarantee the efficiency of maintenance in critical security patching scenarios. Therefore, we introduce the concept of Vulnerability Surface, which aims to aid cloud operators to evaluate the effectiveness of maintenance strategies on minimizing security breaches.

Besides, we present a heuristic algorithm that focuses on improving the trade-off between preserving applications' availability and reducing the data center's Vulnerability Surface. We show that our proposal overcomes well-known strategies in terms of maintenance time, the number of migrations per virtual machine, and the Vulnerability Surface.

1.2 Organization

The remaining of this work is organized as follows:

- Chapter 2 elaborates on the fundamental concepts used in this study, especially regarding some of the main principles of cloud computing and how maintenance is performed in cloud environments.
- Chapter 3 presents other studies that focus on optimizing maintenance in cloud data centers, discusses the different aspects addressed by each of those studies, and indicates how this work complements the literature.
- Chapter 4 presents the main contributions of this thesis, namely: *(i)* the definition of the Vulnerability Surface metric, and *(ii)* a heuristic algorithm for minimizing the Vulnerability Surface during servers maintenance in data centers.
- Chapter 5 is divided into two parts: the first one presents the methodology used for validating the proposed heuristic; the latter shows the evaluation results.
- Finally, Chapter 6 is reserved for the final considerations, which briefly reviews the main contributions of this study and give directions for future research.

2. BACKGROUND AND MOTIVATION

This chapter presents a theoretical background on cloud computing, discussing the main characteristics of this model, and describing how maintenance is performed in cloud environments.

2.1 Cloud Computing

Cloud computing has become widely popular across the Information Technology (IT) industry by allowing multi-sized companies to offer their businesses on a large scale with reduced investments with computing infrastructure [3]. The basic idea of cloud computing consists of delivering computing resources as services remotely over the Internet. According to the National Institute of Standards and Technology (NIST) [18], cloud computing allows its customers the on-demand access to ubiquitous, configurable resources with minimal management effort.

Cloud computing promotes cost reduction by providing third-party managed infrastructure, but also by allowing customers to only pay for the amount of resources that are actually being used [13]. This concept is called "*pay-per-use*" or "*pay-as-you-go*" billing, which replaces fixed fees by specific charges based on computing resources usage. Any additional cost with equipment replacement or maintenance performed in the background by the provider is not charged [11]. In addition to its flexible billing model, Cloud computing is also known through five particular characteristics: *(i)* On-demand self-service; *(ii)* Broad network access; *(iii)* resource pooling; *(iv)* rapid elasticity; and *(v)* measured service.

The first two characteristics regard cloud customers' autonomy in allocating and accessing cloud resources. The on-demand self-service indicates that cloud customers can rent computing resources without any interaction with the cloud vendor. This feature is provided by the cloud platform's autonomic resource orchestration mechanisms that automate the building, deployment, and management of applications. The broad network access characteristic denotes the cloud's interoperability, which allows users to access services from different devices through the Internet.

The third and fourth characteristics regard the cloud's ability to offer virtually infinite resources with transparency regarding physical location. To accomplish this goal, cloud services receive a dynamic slice of resources that can be acquired or released according to the demand generated by customers. The fifth characteristic regards metering the resources provided by cloud computing to provide transparency for both the provider and the customer.

Cloud vendors can also implement different service models to supply the various needs of companies and individuals from different businesses. Cloud service models are meant to fit with a different set of workflows, ranging from low-level provisioning of infrastructure resources such as storage, networks, and servers to high-level ready-to-use environments. According to NIST [18], Cloud computing incorporates three service models:

- *Infrastructure-as-a-Service (IaaS)*: the IaaS service model gives customers the on-demand access to virtually provisioned hardware resources. Therefore, customers can host their platforms on a cloud vendor's infrastructure, saving costs with hardware acquisition and maintenance.
- *Platform-as-a-Service (PaaS)*: the PaaS offering concentrates on delivering pre-configured cloud environments where customers can develop, test, and manage applications for their businesses without the need for handling infrastructure-specific issues.
- *Software-as-a-Service (SaaS)*: the SaaS model gives direct access to ready-to-use vendor's cloud-based applications through the web. Therefore, customers do not have to deal with both infrastructure and platform configuration.

Cloud deployment models are organized according to access restrictions, size, and ownership. In broad terms, it is possible to use cloud services on third-party-owned resources or single-entity resources. The NIST [18] defines four different cloud deployment models:

- *Private clouds*: rely on resources solely owned by a specific organization. This deployment model allows for enhanced security measures such as particular network restrictions. Although a third-party entity can manage private cloud's physical infrastructure, resources are not shared with other individuals.
- *Public clouds*: do not restrict resource usage to a single organization, which means that several companies can share physical resources provided by the same vendor. Although public clouds provide several security layers, there are higher security risks than private clouds as a result of sharing resources.
- *Hybrid clouds*: combine private and public cloud characteristics to meet varying demands. This deployment model usually consists of private environments that may eventually allocate resources from public clouds. Hybrid clouds can be used by companies with highly variable demand so that privately cloud-hosted applications can utilize public cloud functionality when the demand exceeds the limits provided by the private cloud.
- *Community clouds*: cloud environments owned by a group of individuals with common purposes. Customers cut expenses by not having to hold a private cloud but also ensure some levels of access restriction by only sharing resources with members of the institutions that own the community cloud.

Supporting all features expected from cloud applications leverages concerns with maintaining a considerable amount of computing systems that must be kept at strict conditions. To this end, dedicated facilities specially designed to host computer systems are used. These dedicated spaces are called data centers and employ specialized power supplies and cooling mechanisms to support large-scale computing infrastructure. The data center infrastructure has changed from a tightly coupled model where each application was deployed on an individual physical server to a fully virtualized environment, where compute, storage, and network resources are provided on a virtualized fashion.

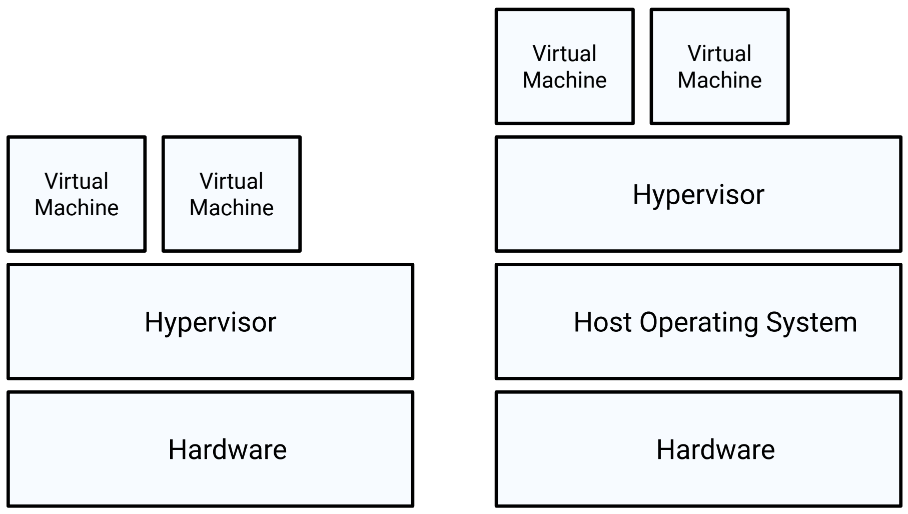
2.2 Virtualization

Nowadays, virtualization is the core technology on providing virtual abstractions for cloud data centers resources [32] [30]. Virtualization utilizes software abstractions to create virtual resources such as virtual components, virtual devices, or even virtual machines.

The virtualization component responsible for creating and managing virtual machines is called hypervisor [29] [1]. One of the hypervisor's main feature consists of creating logically separated environments (the virtual machines) and assigning each a slice of the physical resources. Cloud environments utilize this feature as a means for allowing multitenancy, where multiple applications run on isolated subsystems inside a single physical host for better resource utilization. Figure 2.1 depicts the architecture of the two main categories of hypervisors, namely Type 1 and Type 2 hypervisors. The main differences between these categories are described next:

- *Type 1*: also known as bare-metal architecture, manages guest operating systems (virtual machines) directly from the host's hardware, replacing the host operating system. This type of hypervisor is very efficient in terms of performance and security, as there are no abstraction layers between them and the physical resources that could promote bottlenecks or attacks.
- *Type 2*: also known as hosted architecture, controls virtual machines from inside the host operating system, running as a conventional application. This type of hypervisor's primary goal is providing access to alternative guest operating systems without the need for replacing the host operating system. This virtualization type generates an extra overhead compared to the Type 1 hypervisors. As a consequence, server-based environments usually choose to adopt Type 1 hypervisors.

In addition to hosting multiple applications in the same server, modern cloud data centers also utilize a virtualization technique called migration to move virtual machines between different servers. The migration process consists of transferring memory and storage



(a) Type 1 (Bare-Metal) Architecture. (b) Type 2 (Hosted) Architecture.

Figure 2.1: An overview on the two main hypervisor architectures.

information that constitute the virtual machine state through the network to the destination host server. There are two main techniques for migrating virtual machines: *(i)* cold migration and *(ii)* live migration.

Cold migration consists of suspending the virtual machine on the source host and then transferring the operating system and applications state through the network to the destination host. Some hypervisors minimize cold migration duration when servers share the same storage. In these scenarios, the hypervisor just informs the destination server about the virtual machine disk location.

Live migration technique transfers the virtual machine state between hosts without stopping the operating system and applications. Moving the virtual machine's memory state from the source to the destination host can be performed in two different ways: *(i)* pre-copy memory migration [5] and *(ii)* post-copy memory migration [9].

In the pre-copy memory migration, the hypervisor copies all memory pages from the source to the destination server while the virtual machine still runs on the source. Then, the hypervisor starts an iterative copy process, which sends to the destination server all memory pages that changed during the initial copy. In the post-copy memory migration, the virtual machine is stopped on the source server, and a minimal device state (CPU state, non-pageable memory, etc.) is transferred to the destination. Then, the virtual machine is resumed on the destination server, and the remaining memory pages are moved.

Virtual machine migration is utilized by data center operators to achieve several resource optimization goals, such as: *(i)* stacking virtual machines on a reduced number of servers to reduce power consumption [24] [23], or *(ii)* separating applications with similar resource bounds in order to avoid performance interference [21] [16]. Besides, migration can also be used by operators to move virtual machines during maintenance in order to ensure applications' continuity.

2.3 Maintenance on Cloud Data Centers

As most cloud services demand specific quality of service goals such as high availability and low latency, cloud data center operators perform regular maintenance activities to retain physical and logical components working on expected conditions.

Effective maintenance is capable of avoiding up to 40% of infrastructure failures [34]. However, some maintenance activities may require replacing or restarting components to take effect. Therefore, maintenance also raises concerns regarding service outage [6] [7]. These side effects leverage researches around maintenance planning with the focus on preserving acceptable levels of performance and availability of applications without sacrificing maintenance timeliness [20].

In maintenance scenarios that affect applications functioning, affected virtual machines could be stopped until the maintenance is finished. However, this may raise concerns with availability requirements. Therefore, one alternative would be to migrate virtual machines to other active servers within the data center.

Several events may lead to maintenance in cloud data centers, each one with possibly different requirements. For instance, preventive maintenance may not require a strict completion deadline. On the other hand, maintenance events for correction of security vulnerabilities usually need to be applied immediately to preserve the reliability of applications. Examples of security threats to cloud data centers are numerous:

- *Meltdown [17]:* discovered in 2018, the Meltdown attack takes advantage of unpatched operating systems to breach the isolation between applications and the operating system, allowing an attacker to get access to unauthorized memory positions.

- *Spectre [14]:* also reported in 2018, the Spectre vulnerability exploits branch prediction mechanisms present in modern processors to allow attackers to break the isolation between applications, getting access to sensitive data.

- *Fallout [4]:* reported in late 2019 by breaching even Meltdown-patched CPUs, the Fallout attack exploits the CPU store buffer, used for ensuring applications can write specific cache lines they possess. This attack grants intruders access to recently written data, cryptographic keys from other applications, and other sensitive data.

- *RIDL (Rogue In-Flight Data Load) [25]:* similar to Fallout, RIDL, which was also reported in late 2019, exploits vulnerable CPU's internal buffers to leak data from other processes, from the host operating system, and even from other virtual machines.

3. STATE OF THE ART

As maintenance tasks help to safeguard the health of cloud data centers, several researchers have been focused on developing solutions to improve the effectiveness of maintenance either by ensuring its timeliness or reducing its impact on customers' applications.

3.1 Zheng et al., 2013

Zheng et al. [34] focus on finding the best schedule for maintenance while respecting their deadlines, and the number of users requests. The authors consider a set of scenarios where each server within the data center has a maintenance deadline, and virtual machines are only occupied during a period of time.

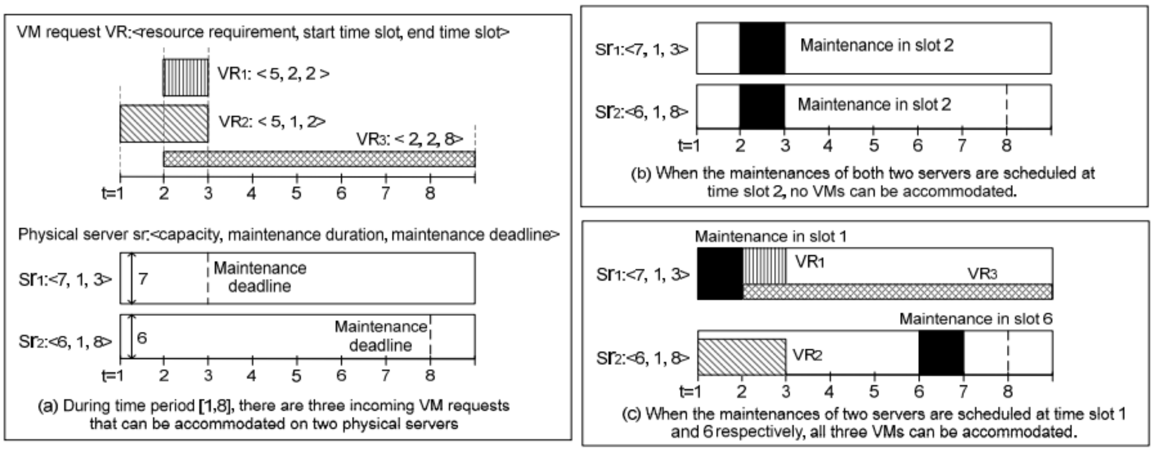

VM request VR:<resource requirement, start time slot, end time slot>
VR1: < 5, 2, 2 >
VR2: < 5, 1, 2>
VR3: < 2, 2, 8>
t=1 2 3 4 5 6 7 8
Physical server sr:<capacity, maintenance duration, maintenance deadline>
sr1:<7, 1, 3>
7
Maintenance deadline
sr2:<6, 1, 8>
6
Maintenance deadline
(a) During time period [1,8], there are three incoming VM requests that can be accommodated on two physical servers
Maintenance in slot 2
Maintenance in slot 2
(b) When the maintenances of both two servers are scheduled at time slot 2, no VMs can be accommodated.
Maintenance in slot 1
VR1
VR3
Maintenance in slot 6
VR2
(c) When the maintenances of two servers are scheduled at time slot 1 and 6 respectively, all three VMs can be accommodated.

Figure 3.1: Virtual machines provisioning and maintenance scheduling strategy proposed by Zheng et al. [34].

The authors propose an approach that takes advantage of virtual machines inactivity period to consolidate them inside a sub-set of servers which were already updated or have a distant maintenance deadline. As depicted in Figure 3.1, the authors show that considering both the virtual machines' occupation period and the servers maintenance deadlines allow operators to schedule servers for maintenance even in scenarios with limited resource capacity.

3.2 Yanagisawa et al., 2013

Yanagisawa et al. [31] argue that one of the most outstanding challenges in allocating virtual machines during maintenances is related to meeting the demand of the virtual machines being migrated from servers scheduled to maintenance without compromising fault-tolerance schemes.

The authors present a mixed-integer programming approach that performs server maintenance while preserving a fault-tolerance mechanism called Active-Standby (depicted in Figure 3.2). The Active-Standby fault-tolerance mechanism consists of creating a replica that is kept inactive for each virtual machine inside the data center. The general idea of the fault-tolerance maintenance scheme proposed in this study consists of avoiding migrating all virtual machine replicas to the same server. If a failure occurs with one of the virtual machines or a server stops working, the standby replica is activated in order to preserve the application availability.

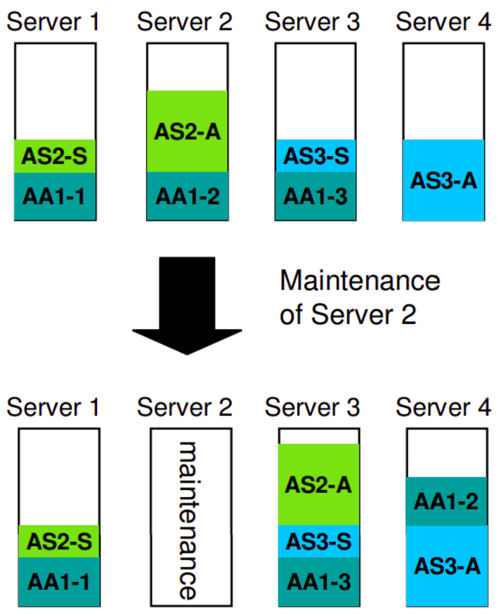


Figure 3.2: Dependability-aware maintenance strategy proposed by Yanagisawa et al. [31].

3.3 Zheng et al., 2014

Zheng et al. [35] discuss that preserving acceptable levels of availability during maintenance in cloud data centers is one of the most challenging aspects for cloud operators since if all servers are scheduled for maintenance at the same time, all applications will suffer from service outage. On the other hand, performing the maintenance of servers one by one may not be acceptable in cases of critical updates.

The authors present a heuristic that tries to maximize the number of servers to be updated at the same time without compromising applications' availability. When selecting servers to be emptied, the proposed heuristic prioritizes the less occupied ones, which leads to more servers being updated at the same time. Besides, authors avoid unnecessary migrations by prioritizing the migration of virtual machines to servers already updated. An example of the proposed heuristic is presented in Figure 3.3.

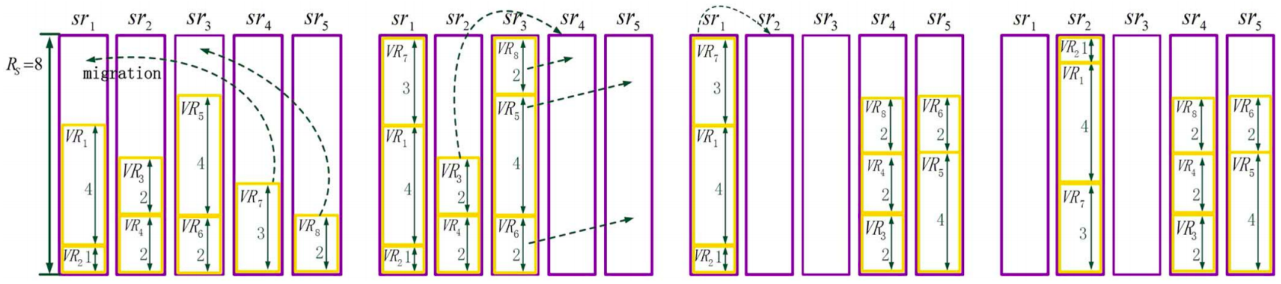


Figure 3.3: Batch scheduling scheme for server maintenance proposed by Zheng et al. [35].

3.4 Wang et al., 2014

Wang et al. [27] argue that maintenance scenarios that demand restarting servers to perform the update bring operators the decision of whether migrating virtual machines or temporarily stopping them if they are not receiving user requests.

On the one hand, migrating virtual machines increases the data center's network and memory usage. On the other hand, the downtime caused by stopping virtual machines may affect the users' experience. Therefore, the authors present a heuristic that chooses between migrating virtual machines (migration cost) and temporarily stopping them during the maintenance period (delay of waiting for the server to be updated). An example of how the proposed heuristic makes its decisions is depicted in Figure 3.4.

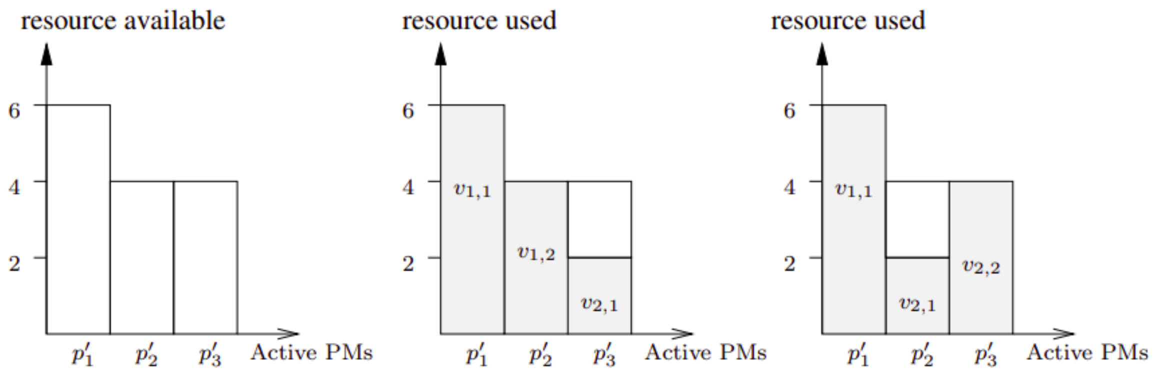
The authors improve the objective function's flexibility using a variable α, which denotes the balance between the delay and the migration cost. As the value of α increases, the more prominent will be the impact of the migration cost on the objective function. For example, if $\alpha \leftarrow 0$, then the objective function will seek only minimizing the delay (by migrating all virtual machines). On the other hand, if $\alpha \leftarrow \infty$, then the objective function will pay more attention to minimize the migration cost by stopping virtual machines on their original hosts instead of migrating them before starting the maintenance.

3.5 Okuno et al., 2019

Okuno et al. [19] focus on minimizing the impact caused by maintenance on applications by scheduling the migration of virtual machines while preserving maintenance timeliness. The authors achieve this goal by using a divide-and-conquer approach that divides the maintenance scheduling problem into a set of sub-problems, which are solved individually, and then combined into a solution for the initial problem.

Maintained PMs	Maintained duration	VMs located in maintained PMs		Migration cost between PMs		
		VMs	Resource required by each VM	p'_1	p'_2	p'_3
p_1	3	$v_{1,1}$	6	2	2	2
		$v_{1,2}$	4			
p_2	2	$v_{2,1}$	2	2	1	1
		$v_{2,2}$	4			

(a) VMs located at PMs and migration cost information



(b) Resource available at active PMs (c) The first scheme with delay 2 and migration cost 5 (d) The second scheme with delay 3 and migration cost 4

Figure 3.4: Cost-Delay aware maintenance scheme proposed by Wang et al. [27].

The proposed strategy seeks to create groups of servers to be updated simultaneously. To create these groups, the proposed solution looks for combinations that respect three conditions: *(i)* all virtual machines hosted by the servers within the group must be migrated in at least one time slot; *(ii)* different instances of the same application must not be migrated at the same time to respect the anti-affinity rules; and *(iii)* capacity constraints must not be violated. Once the groups of servers to be updated simultaneously are defined, the proposed solution updates them all in parallel. This process is depicted in Figure 3.5.

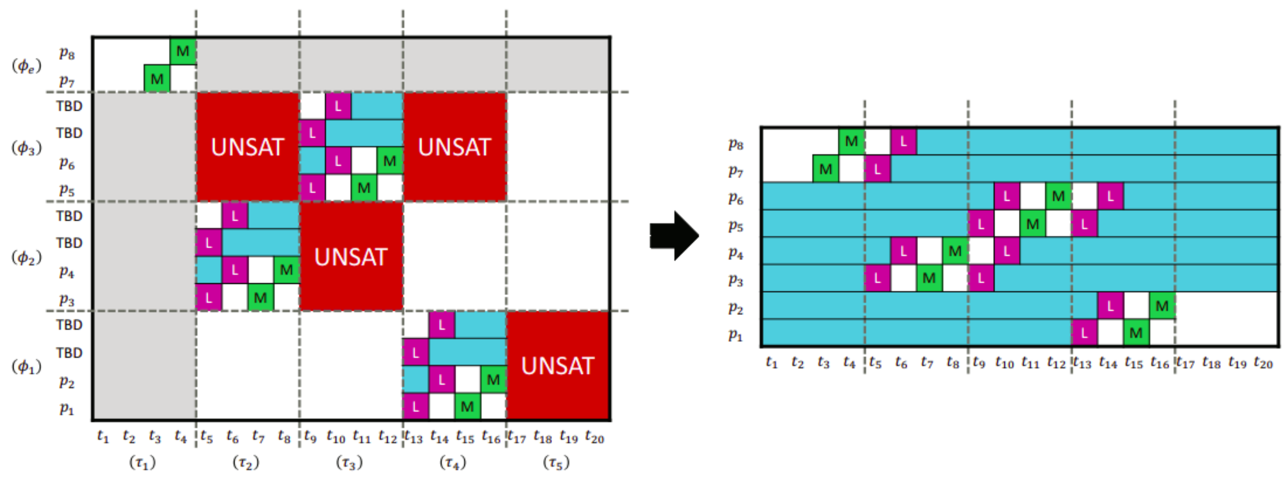

(ϕ_e)
(ϕ_3)
(ϕ_2)
(ϕ_1)
p_8
p_7
TBD
p_6
p_5
p_4
p_3
p_2
p_1
M
L
UNSAT
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16} t_{17} t_{18} t_{19} t_{20}
(τ_1)
(τ_2)
(τ_3)
(τ_4)
(τ_5)

Figure 3.5: Maintenance scheduling scheme proposed by Okuno et al. [19].

3.6 Discussion

In this section, we discuss the different goals considered by previous investigations, and then we elaborate on how this study complements the existing literature.

As we can see in Table 3.1, there is a broad spectrum of optimization goals that can be considered during server maintenance in cloud environments. Some of these objectives achieve improvements that can be felt by end-users directly. For example, minimizing the number of migrations performed throughout the maintenance could avoid performance bottleneck and downtime perceived by end-users accessing applications during the maintenance. Amongst the related studies cited above, we can mention the example of Zheng et al. [34], which focuses primarily on scheduling maintenance in idle periods as a means for minimizing the impact of maintenance on applications.

Apart from the enhancements that impact end-user experience directly, such as reduced downtime, several decisions could be made during server maintenance to benefit users indirectly. As an example, we can deliberate on Yanagisawa et al. [31], which focuses on ensuring that maintenance migrations will not affect fault-tolerance schemes' effectiveness. Even though fault-tolerance schemes do not intend to provide gains immediately visible to end-users, such preventive measures end up by allowing applications to deliver a higher quality of service long term.

Given the increasing concern with attacks against cloud servers, finding ways to update servers as soon as possible is mandatory. Even though current studies present solutions for optimizing maintenance direct and indirectly, none of them concentrate on security patching scenarios. Therefore, this study aims at complementing the existing literature by helping operators to minimize the amount of time servers stay exposed to attacks while waiting for update during server maintenance.

Table 3.1: Summary of the main goals of related studies.

Work	Primary Goal
Zheng et al. [35]	Reducing maintenance duration
Okuno et al. [19]	
Yanagisawa et al. [31]	Preserving fault tolerance
Wang et al. [27]	Optimizing virtual machine management
Zheng et al. [34]	Minimizing the impact of maintenance on applications
This Work	Safeguarding servers against attacks

4. HEURISTIC ALGORITHM FOR SERVER MAINTENANCE

This chapter presents the proposed work of server maintenance in cloud data centers focused on critical security patching scenarios. This chapter is divided into four parts:

- Initially, the motivation of this work is presented, which discusses some challenges of performing maintenance in critical security scenarios.
- The second part of this chapter presents the Vulnerability Surface metric, which aids operators to evaluate maintenance strategies on critical security patching scenarios.
- The third part of this chapter presents a detailed description of server update during critical security maintenance in cloud data centers.
- Finally, a heuristic algorithm that solves the problem is described.

4.1 Motivation

Effective maintenance is essential for preserving cloud data centers on operable conditions. As highly connected devices, cloud servers have several points of failure. Possible threats range from unusual network traffic intended to overload servers capacity to malicious applications trying to breach the host operating system isolation to have access to data from other applications. Therefore, effective maintenance planning comes on the scene to mitigate the effects of these threats through several possible actions such as upgrading operating system kernel, applying hypervisor hotfixes, or even replacing defective components.

Despite the benefits of maintenance on the overall data center's dependability, several maintenance activities require rebooting servers to take effect. Previous studies investigate the possibility of delaying or advancing maintenance to time windows wherein applications are receiving fewer requests so that the impact caused by the downtime will be reduced [34]. However, in some urgent scenarios delaying maintenance until the users' demand reduces may not be an alternative. In these cases, operators must select some servers to be updated at the same time, and then reallocate applications from these servers before starting the update. Related studies that propose solutions for similar scenarios focus only on minimizing the number of migrations and, therefore, minimizing the overall maintenance time.

Although minimizing the overall maintenance time increases the maintenance effectiveness in emergency scenarios, it does not necessarily mean that the maintenance strategy efficiently manages to secure the largest number of servers as possible. For instance, in Figure 4.1 we present two maintenance strategies, namely Consolidation-Aware Strategy and Delay-Aware Strategy, for updating 3 servers (with 10 units of capacity each). The Consolidation-Aware Strategy focuses on migrating as many virtual machines as possible to consolidate the workload on a minimum number of servers, based on the premise that this decision will eventually lead to more servers empty for maintenance. The Delay-Aware Strategy only migrates virtual machines that will immediately (i.e., in the same maintenance step) lead to a new empty server.

For simplicity purposes, we consider sequential migrations. For the migration delay, we consider both the time for saving and restoring the virtual machine as 10 seconds and the network delay as twice the size of the virtual machine. Therefore, the migration cost for a given virtual machine vm_j is set to 10 + $vm_{j,size} \times 2 + 10$, and the server update time is set to 60. Both strategies take the same amount of time to update Servers 1, 2, and 3, but expose them to attacks during different periods. In this scenario, the main difference between the two strategies can be observed in Step 2, wherein the Consolidation-Aware Strategy chooses for consolidating virtual machines as much as possible, without discarding migrations that will not result in a new empty server.

The additional cost of migrating the virtual machine VM4 from Server 2 to Server 3 causes an extra delay on the migration period of the Consolidation-Aware Strategy in Step 2, which leads to 160 seconds where the data center has 2 servers vulnerable to attacks. On the other hand, the approach adopted by the Delay-Aware Strategy, which chose for not migrating this virtual machine, led to a faster server update, wherein the data center had 2 vulnerable servers for only 134 seconds (26 seconds less than the Consolidation-Aware Strategy).

In this study, we argue that only focusing on minimizing the maintenance duration and preparing for maintenance as many servers as possible similar to the related studies does not guarantee the effectiveness of maintenance strategies on critical security patching scenarios, wherein securing servers against attacks is the greatest priority.

It is worth bearing in mind that both strategies in the previous example take the same amount of time and require the same number of virtual machine migrations to perform the maintenance. This indicates that only considering maintenance time, number of migrations, and number of updated servers simultaneously as performance metrics may not be sufficient to assess the efficiency of a maintenance strategy on a critical security maintenance scenario. Therefore, in this study, we introduce the concept of Vulnerability Surface, which aims to aid operators in measuring how efficiently maintenance strategies minimize security breaches.

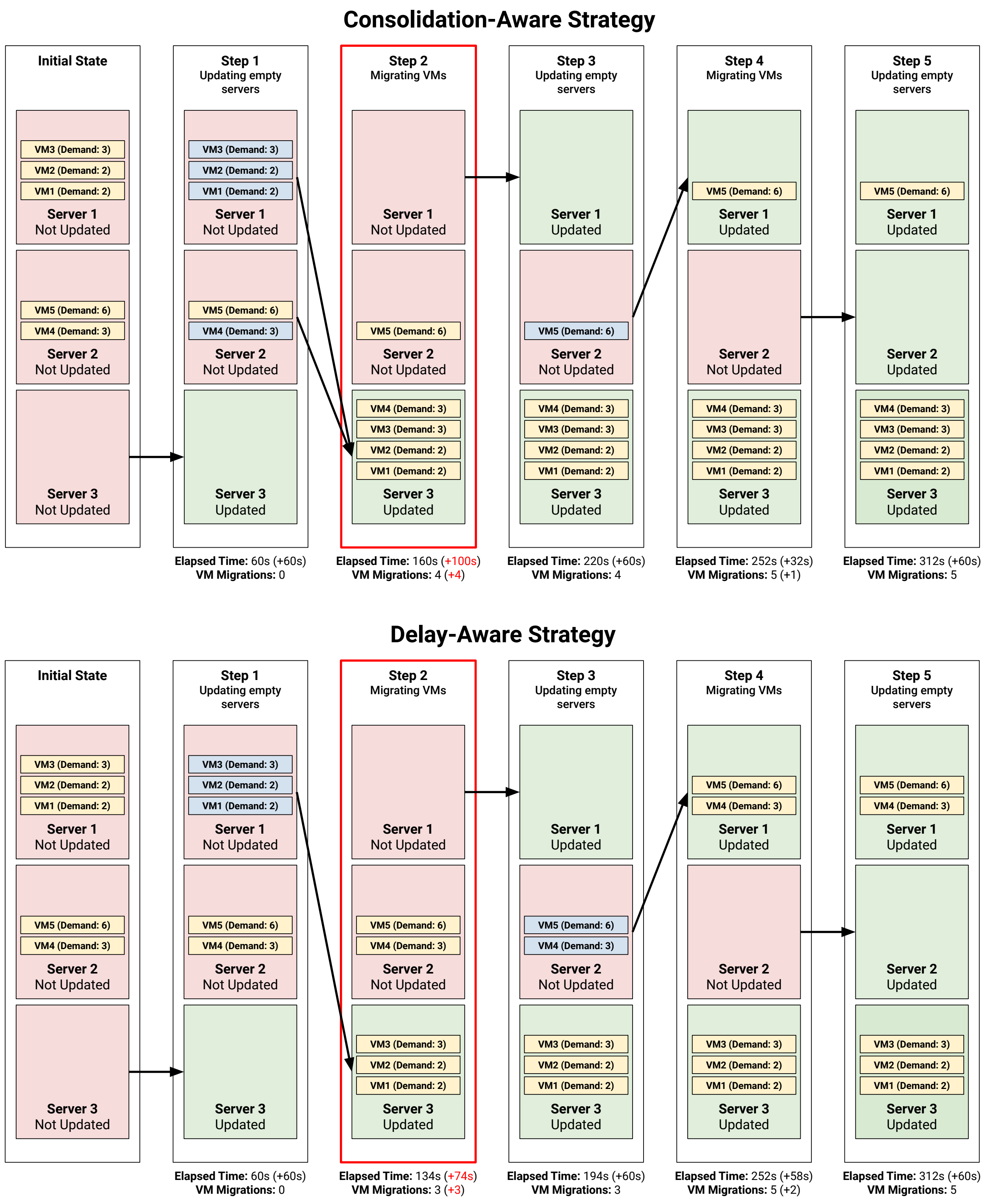

Consolidation-Aware Strategy
Initial State
Step 1 Updating empty servers
Step 2 Migrating VMs
Step 3 Updating empty servers
Step 4 Migrating VMs
Step 5 Updating empty servers
VM1 (Demand: 2)
VM2 (Demand: 2)
VM3 (Demand: 3)
VM4 (Demand: 3)
VM5 (Demand: 6)
Server 1 Not Updated
Server 2 Not Updated
Server 3 Not Updated
Server 1 Updated
Server 2 Updated
Server 3 Updated
Elapsed Time: 60s (+60s) VM Migrations: 0
Elapsed Time: 160s (+100s) VM Migrations: 4 (+4)
Elapsed Time: 220s (+60s) VM Migrations: 4
Elapsed Time: 252s (+32s) VM Migrations: 5 (+1)
Elapsed Time: 312s (+60s) VM Migrations: 5
Delay-Aware Strategy
Elapsed Time: 60s (+60s) VM Migrations: 0
Elapsed Time: 134s (+74s) VM Migrations: 3 (+3)
Elapsed Time: 194s (+60s) VM Migrations: 3
Elapsed Time: 252s (+58s) VM Migrations: 5 (+2)
Elapsed Time: 312s (+60s) VM Migrations: 5

Figure 4.1: How maintenance decisions may produce different levels of the data center's safety during critical security maintenance patching scenarios.

4.2 Vulnerability Surface

The Vulnerability Surface seeks to support the evaluation of maintenance strategies regarding how efficient each maintenance decision helps to safeguard cloud servers against security threats. The Vulnerability Surface determines the efficiency of a maintenance strategy based on the amount of time that servers need to wait before being updated (i.e., the period they stay vulnerable to attacks).

Let $\mathbb{T} \leftarrow \{t_1, t_2, ..., t_n\}$ be the set of iterations performed to update a cloud data center. The elapsed time at the end of each iteration t_n is stored in a vector Δ_n and the number of vulnerable servers (i.e., not updated servers) is stored in $\varphi(n)$. The Vulnerability Surface classifies the efficiency of a maintenance strategy based on the number of not updated servers at the end of each iteration $t_i \in \mathbb{T}$. Bearing in mind that in critical security maintenance scenarios we seek to update servers as soon as possible to safeguard them against attacks. We define the Vulnerability Surface as:

$$\Delta \cdot \varphi \tag{4.1}$$

The Vulnerability Surface presented in Equation 4.1 seeks to assess maintenance strategies according to how fast they update servers. Therefore, the faster a maintenance strategy update servers, the lower will be its Vulnerability Surface.

Existing approaches only focus on consolidating virtual machines in order to let servers prepared for maintenance as soon as possible. Differently, the Vulnerability Surface considers the trade-off between migrating virtual machines to empty servers and the amount of time required by the migration. Therefore, strategies that avoid migrations that not lead to a new server being emptied get a higher score. Considering the maintenance example presented in Section 4.1, the Vulnerability Surface of both strategies will be defined as follows (for clarity purposes, we present an expanded notation for the Vulnerability Surface's summation):

- **Consolidation Aware:** $\overbrace{2 \times 60}^{\textbf{Step 1}} + \overbrace{2 \times 160}^{\textbf{Step 2}} + \overbrace{1 \times 220}^{\textbf{Step 3}} + \overbrace{1 \times 252}^{\textbf{Step 4}} + \overbrace{0 \times 312}^{\textbf{Step 5}} = 912$
- **Delay Aware:** $\overbrace{2 \times 60}^{\textbf{Step 1}} + \overbrace{2 \times 134}^{\textbf{Step 2}} + \overbrace{1 \times 194}^{\textbf{Step 3}} + \overbrace{1 \times 252}^{\textbf{Step 4}} + \overbrace{0 \times 312}^{\textbf{Step 5}} = 834$

Until the second step, both strategies had the same Vulnerability Surface. However, the choice made by the Consolidation-Aware Strategy of migrating the virtual machine VM4 to Server 3 negatively impacted the overall score of this strategy since this migration delayed the server update performed in the next step and did not lead to a new empty server.

4.3 Problem Formulation

In this work, we consider a cloud data center scenario containing several physical servers. Each physical server may host one or several virtual machines that accommodate applications. In such a situation, all physical servers within the data center must undergo maintenance as soon as possible while ensuring applications' availability.

Let $\mathbb{S} = \{S_1, S_2, ..., S_i\}$ be the set of i physical servers within the data center. Each physical server has a resource vector that represents CPU, memory, and disk capacity. As stated in the previous chapters, countless kinds of maintenance could be performed to retain an operable condition in cloud data centers. In this study, we focus on server maintenance events with strict completion deadline, which also require the server to be restarted after the update. Therefore, we state that servers must be emptied before undergoing maintenance in order to preserve applications' continuity.

The set of virtual machines accommodated in the physical servers is represented by $\mathbb{V} = \{VM_1, VM2, ..., VM_j\}$, in which every virtual machine has a resource vector that represents its CPU, memory, and disk demands. We represent the initial placement of virtual machines on physical servers with the following binary matrix:

$$x_{i,j} = \begin{cases} 1 & \text{if } PM_i \text{ is hosting } VM_j \\ 0 & \text{otherwise.} \end{cases}$$

In this study, we divide the maintenance process into one or several iterations of two events: (i) **virtual machines migration**, performed before updating the physical servers in order to preserve applications' continuity; and (ii) **servers maintenance**, wherein servers emptied in the previous step are updated. For the migration process, we consider cold migration, wherein the virtual machine state is saved at the host server, this state is migrated throughout the network, and when the transferring is completed, the state is restored at the destination server. Given a scenario with no shared storage, we also consider migrating the virtual machine disk. Therefore, in this study, we calculate the migration time of a virtual machine vm_j as the following equation:

$$migrationTime(vm_j) = saveState(vm_j) + \left(\left(\frac{vm_{j,diskSize} + vm_{j,memorySize}}{Network\ Bandwidth}\right)\right) + restoreState(vm_j)$$

Considering such a maintenance scenario, our goal is threefold: *(i)* maximizing the data center's security in face of critical security maintenance by reducing the Vulnerability Surface; *(ii)* maximizing the number of physical servers updated simultaneously in order to reduce the maintenance time; and *(iii)* minimizing the number of migrations per virtual machine to reduce the impact of maintenance on applications performance.

4.4 Proposed Heuristic

As mentioned in the previous sections, in this work, we consider the cloud servers maintenance process as an iteration of two processes: *(i)* migrating virtual machines in order to empty servers and *(ii)* updating emptied servers. The process of migrating virtual machines is a variant of the Vector Bin Packing problem, which is an NP-Hard problem [10] [22].

Given a set $S \leftarrow \{S_1, S_2, ..., S_n\}$ of n bins, each with a pre-defined capacity $\sigma(S_i)$, and a set $A \leftarrow \{A_1, A_2, ..., A_j\}$ of j items with size $\rho(A_j)$, the Bin-Packing problem seeks at finding a given arrangement of items into bins that minimizes the number of bins necessary for accommodating all items without exceeding bins capacity.

The Bin-Packing problem can be applied to represent real-world problems of several areas: saving investments with logistics by minimizing the number of containers necessary for transporting products, increasing TV station incomings by arranging advertisements in fixed-length TV show breaks, and so on.

The Bin-Packing problem also presents several variants, such as the Vector Bin Packing, in which each item size is represented by a N-dimensional vector. The Vector Bin Packing Problem can be applied during server maintenance in cloud data centers, where choosing the best arrangement of virtual machines inside physical resources may lead to more empty hosts available for maintenance.

Given the NP-hardness of the Vector Bin Packing problem, as the number of considered elements increases, the search space increases drastically, which makes prohibitive the use of optimization techniques that guarantee an optimal solution, such as linear programming. For this reason, in this study, we present a heuristic algorithm that can ensure acceptable solutions in a bounded time.

The proposed solution for minimizing server maintenance and Vulnerability Surface in cloud data centers is presented in Algorithm 4.1. The list of variables and functions used in the algorithm is presented in Table 4.1. The proposed algorithm is divided into two distinct phases: *(i)* servers maintenance and *(ii)* virtual machines migration. The maintenance phases that constitute the proposed heuristic algorithm are described in the next sections.

4.4.1 Servers Maintenance

The first phase aims at updating servers that are not hosting virtual machines. In this phase, depicted in lines 2-5 of Algorithm 4.1, the proposed heuristic creates a subset of servers $\mathbb{E} \in \mathbb{P}$, which is occupied by not updated servers that host no virtual machines. Then, the algorithm iterates over $\mathbb{E}$, performing the maintenance on the empty servers.

Table 4.1: List of parameters used on the proposed heuristic.

Parameter	Description
$\mathbb{S}$	Set of servers in the data center
$\mathbb{V}$	Set of virtual machines in the data center
$\mathbb{E}$	Set of servers ready for maintenance (already empty)
$\mathbb{U}$	Set of updated servers
$\mathbb{Y}$	Set of servers being prepared for maintenance
$\mathbb{M}$	Set of not updated servers
$\mathbb{H}$	Collection of virtual machines that need to be migrated in order to set a server as ready for maintenance
$\mathbb{P}$	List of destination server candidates for a virtual machine that needs to be migrated
$\mathit{updated}(s_i)$	Binary function that returns 1 if a server s_i is updated, and returns 0 otherwise
$\mathit{maintenance}(s_i)$	Function that performs the maintenance on a server s_i
$\mathit{capacity}(s_i)$	Function that returns the capacity of a server s_i
$\mathit{demand}(s_i)$	Function that returns the demand of a server s_i
$\mathit{demand}(vm_i)$	Function that returns the demand of a virtual machine vm_j
$\mathit{freeResources}(s_i)$	Function that returns the amount of resources available on a server s_i
$\mathit{migrate}(vm_j, s_k)$	Function that migrates a virtual machine vm_j to a server s_k
$\mathit{updateCost}(s_i)$	Function that returns the update cost of a server s_i based on the amount of time necessary for migrating all virtual machines that it hosts
$\mathit{sort}(\mathit{set}, \mathit{sorting\ property}, \mathit{order})$	Function used for ordering a collection of elements according to a given sorting property and a sorting order (i.e., ascending or descending)

4.4.2 Virtual Machines Migration

The second phase of the algorithm focuses on migrating virtual machines among the available servers to empty servers. This phase is represented by lines 6-23. As the algorithm tries to minimize the data center's Vulnerability Surface, in this step, denoted in the line 6 of Algorithm 4.1, the algorithm prioritizes emptying servers with lower update cost, i.e., that will take less time to be emptied. A server may require a shorter time to be emptied by having a small number of virtual machines or by hosting small virtual machines, which will take a negligible time to be migrated. Once the list of servers to be emptied is formed, the algorithm iterates over each server of this list in order to migrate virtual machines. The proposed heuristic adopts a First-Fit Decreasing strategy, that, given a list of candidate servers, it chooses the first one with enough resources to host each of the virtual machines, starting from the largest one to the smaller one. The order in which candidate servers are organized defines the chances of servers being chosen for hosting a virtual machine.

The proposed heuristic prioritizes updated servers (line 12 of Algorithm 4.1) to avoid unnecessary migrations in the future steps, as all virtual machines migrated to not updated servers will have to be relocated at least one more time to empty its host server. Besides, updated servers with higher occupation rates are put first in the set, as choosing less empty servers to host virtual machines may result in better packing of virtual machines, which in turn may allow more virtual machines being migrated per step. Given that updated

Algorithm 4.1 Proposed heuristic for performing data center maintenance.

```
Input: $\mathbb{S} \leftarrow \{S_1, S_2, ..., S_i\}$
       $\mathbb{V} \leftarrow \{VM_1, VM_2, ..., VM_j\}$
 1: while $\sum_i^{\mathbb{S}} updated(s_i) < |\mathbb{S}|$ do
 2:   $\mathbb{E} \leftarrow$ not updated servers $\in \mathbb{S}$ with no VMs
 3:   for each server $s_i \in \mathbb{E}$ do
 4:     $maintenance(s_i)$
 5:   end for
 6:   $\mathbb{M} \leftarrow sort($not updated servers $\in \mathbb{S}, updateCost(), ascending)$
 7:   $\mathbb{Y} \leftarrow \{\}$
 8:   $\mathbb{U} \leftarrow$ updated servers $\in \mathbb{S}$
 9:   for each server $s_i \in \mathbb{M}$ do
10:     $\mathbb{H} \leftarrow sort($virtual machines hosted by $s_i, demand(), descending)$
11:     for each virtual machine $vm_j \in \mathbb{H}$ do
12:       $\mathbb{P} \leftarrow sort(\mathbb{U}, demand(), descending) \cup sort(\mathbb{M} - \{s_i \cup \mathbb{Y}\}, demand(), descending)$
13:       if $\sum_l^{\mathbb{P}} capacity(s_l) \geq demand(s_i)$ then
14:         for each server $s_k \in \mathbb{P}$ do
15:           if $freeResources(s_k) \geq demand(vm_j)$ then
16:             $migrate(vm_j, s_k)$
17:             break
18:           end if
19:         end for
20:         $\mathbb{Y} \leftarrow \mathbb{Y} \cup \{s_i\}$
21:       end if
22:     end for
23:   end for
24: end while
```

servers may not have enough resources to host virtual machines or data centers may not even have updated servers in a given moment, the proposed heuristic also considers the migration of virtual machines to not updated servers. However, it puts them at the end of the list, also ordered by occupation. It is worth bearing in mind that the list of not updated servers that is appended to $\mathbb{P}$ excludes s_i since it cannot migrate its virtual machines to itself, and the servers included in $\mathbb{Y}$, which accommodates all servers emptied in the current step and are waiting for the next iteration to be updated. This constraint avoids swapping virtual machines from servers being emptied.

As depicted in the previous sections, one of the main aspects that may increase the Vulnerability Surface during server maintenance is migrating virtual machines that will not effectively lead to a new server being emptied in the current maintenance step. Therefore, before starting migrating the virtual machines from a server, in the line 13 of Algorithm 4.1, the proposed algorithm checks if there is space within the other servers to host all virtual machines from the server being emptied. It is also worth bearing in mind that a better virtual machine allocation may potentially result in more servers being updated simultaneously. Therefore, the algorithm uses a First-Fit Decreasing strategy to migrate virtual machines, starting from the larger ones to the smaller ones, until all virtual machines from a server were migrated.

As depicted in Figure 4.2, the proposed heuristic minimizes the Vulnerability Surface rate in 5.75% compared to the Delay-Aware Strategy. The reason for such improvement results from the decision to start emptying servers that host a set of virtual machines that demand less time to be migrated. In this example, the Delay-Aware Strategy chooses to migrate the virtual machines from Server 1 to Server 3. In fact, the overall demand of Server 1 is smaller than that from Server 2. However, the additional cost of saving and restoring the state of the three virtual machines from Server 1 (VM1, VM2, and VM3) makes the overall migration time necessary for emptying this server longer than the time needed for emptying Server 2, wherein the saving and restoring processes would have to be performed only twice.

4.5 Final Remarks

During critical security maintenance in cloud data centers usually requires server update as soon as possible since any delay may represent an opportunity to attacks. In this chapter, we argue that only reducing the overall maintenance time does not necessarily guarantee that servers are being safeguarded as soon as possible.

Therefore, we introduce a new metric called Vulnerability Surface that addresses this gap by allowing operators to assess how efficient maintenance strategies protect servers on these scenarios. Then, we present a heuristic algorithm that focuses on minimizing the Vulnerability Surface with minimal overhead. In the next chapter, we present a set of experiments performed to validate the proposal.

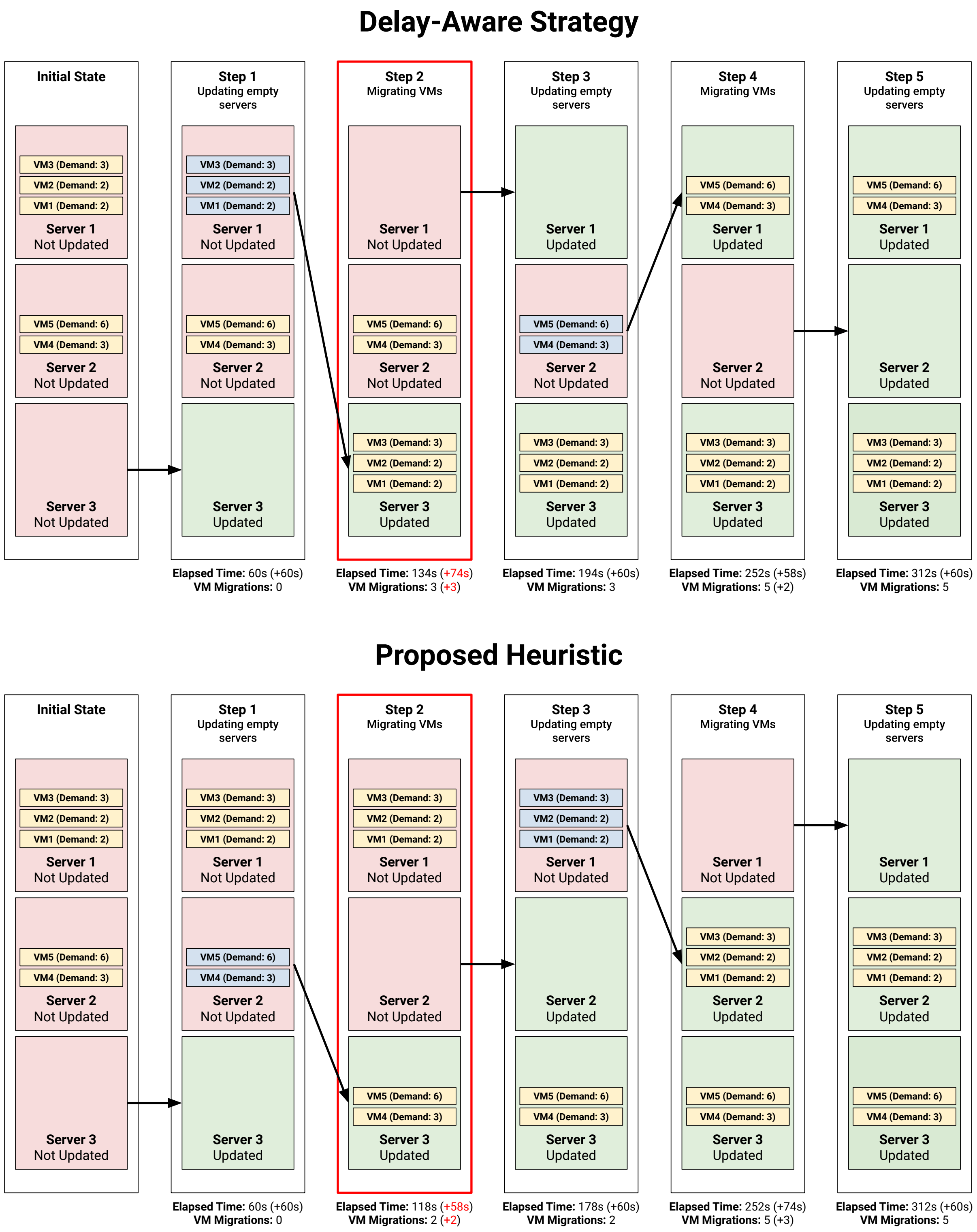


Figure 4.2: How the proposed heuristic minimizes the Vulnerability Surface by prioritizing the maintenance of servers whose virtual machines demand less time to be migrated.

5. EVALUATION AND DISCUSSION

This chapter presents an evaluation carried out to validate the proposed heuristic on critical security patching scenarios on cloud data centers. Initially, the chapter presents a general description of the considered scenarios, including a discussion on the workload characteristics and evaluated solutions. The remainder of the chapter presents and discusses the achieved results.

5.1 Experiments Description

Three workloads, divided by data center's occupation rate, were used to evaluate the effectiveness of the proposed heuristic: *(i) Small Occupation* (25%), *(ii) Medium Occupation* (50%), and *(iii) High Occupation* (75%). Each workload consists of a variable number of physical servers used to host a fixed number of 100 virtual machines. For each server, we consider CPU, memory, and disk capacities. A detailed description of servers capacity is depicted in Table 5.1.

Table 5.1: Server configurations considered during the evaluation.

Type	CPU (# of Cores)	Memory (GB)	Disk (GB)
Small	4	4	32
Medium	8	8	64
Large	16	16	128

We choose for varying the number of servers instead of virtual machines since we use a real-world based virtual machines distribution provided by a 2nd Watch's report[1], that shows the most popular configurations among Amazon Web Services customers during July-October, 2014. Based on this information, virtual machines are divided by capacity, wherein 43% of them are small, 19% medium, and 38% large. The demand of each of these virtual machine configurations is depicted in Table 5.2.

Table 5.2: Virtual machine specifications adopted for the evaluation of the proposed heuristic.

Type	CPU (# of Cores)	Memory (GB)	Disk (GB)
Small	1	1	8
Medium	2	2	16
Large	4	4	32

We consider a cloud data center scenario with a Fat-Tree network topology [15] pre-configured with a per-server bandwidth limit of 1 Gbit/s defined for quality of service

[1] https://www.2ndwatch.com/blog/2nd-watch-aws-scorecard/.

and network fairness purposes. This means that a server has effectively 1Gbit/s of network bandwidth to communicate with any other server within the data center. The evaluated scenarios do not allow for simultaneous virtual machine migrations as a means to ensure the QoS for the remaining applications that rely on networks and hosts used to migrate virtual machines. Therefore, virtual machines are migrated one at a time. Regarding the time required for updating servers, we consider a static period of 180 units of time (defined without loss of generality). All evaluated workloads and heuristics used in this study were created using the Python language. The initial placement for virtual machines is based on a Random-Fit strategy, wherein servers may or not host virtual machines. We use a seed value as input for the workload generator in order to enable reproducibility. Using the same seed, we generate the three scenarios depicted earlier, based on different data center's occupation rates.

As mentioned in Section 4.4, in the context of this study, the server maintenance process has a high computational complexity as migrating virtual machines among servers is required for preparing servers for maintenance. For this class of problems, solutions that ensure the optimal solution become prohibitively expensive. Therefore, in this work, we focus on evaluating heuristic algorithms for performing the migration decisions that return valid solutions within a bounded time. A description of the considered solutions is presented next.

- **First Fit (FF):** performs a sequential search among the physical servers and selects the first one with sufficient resources to host the virtual machine.
- **Best Fit (BF):** migrates the virtual machine to the physical server that has fewer resources available but still has enough resources to meet the virtual machine demand.
- **Worst Fit (WF):** selects the physical server with more computational resources available that has sufficient resources to host the virtual machine.

The goal of all experiments is to indicate how each algorithm behaves on different critical security patching scenarios on cloud data centers. We evaluated the solutions regarding maintenance time, the total number of migrations, the number of migrations per virtual machine, Vulnerability Surface. All experiments assets, including the source-code for the heuristics and the simulator, are publicly available at our GitHub repository (https://github.com/GRIN-PUCRS/cloud-simulator). In the remainder of this chapter, we analyze how several decisions affect a solution's effectiveness. It is also intended to verify if the proposed heuristic is able to provide good results regarding the considered metrics. All solutions were executed in a Linux host machine whose specifications are depicted in Table 5.3.

Table 5.3: Testbed specifications.

Component	Specification
Processor	Intel Core i7-8650U CPU @ 1.90GHz
Memory	16GB DDR4 @ 2400MHz
Storage	ADATA SU810NS38 SATA 256GB
Operating System	elementary OS 5.1.5 Hera (kernel v5.3.0-59-generic)
Python Interpreter	3.6.9 (default, Apr 18 2020, 01:56:04)

5.2 Analysis on Low Occupation Scenario

The goal of this analysis is to examine the implications of decisions made by maintenance strategies on data center scenarios with a low occupation rate (25%). In this scenario, we consider 135 heterogeneous servers (45 servers per configuration) hosting 100 virtual machines. In addition to the low occupation rate, this scenario distinguishes from the others by having 35 initially emptied servers. Therefore, it is also intended to verify the impact of having initially emptied servers on the maintenance process.

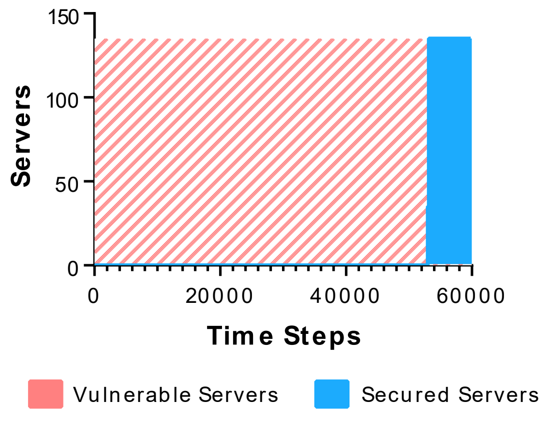
The results obtained executing the solutions are presented in Table 5.4. It is possible to observe that the proposed heuristic significantly overcomes the other strategies on all evaluated metrics.

Table 5.4: Experiment results for the low occupation scenario.

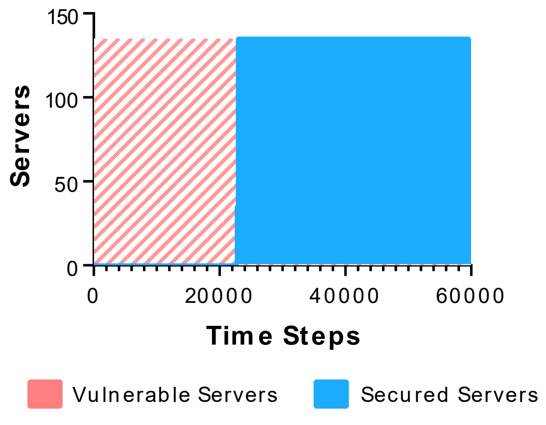
Metrics	Strategy			
	Best-Fit	First-Fit	Worst-Fit	Proposed Heuristic
Maintenance Time	53158	22889	26410	19464
Virtual Machine Migrations	276	117	143	100
Vulnerability Surface	5297800	2270900	2623000	1928400

It is possible to observe that the proposed heuristic significantly outperforms the other strategies on all evaluated metrics. Avoiding migrating virtual machines from servers that will not lead to an immediately emptied server is the main factor that contributed to the gains achieved by the proposed heuristic. As each virtual machine migration is performed sequentially, performing unnecessary migrations significantly increases the maintenance time.

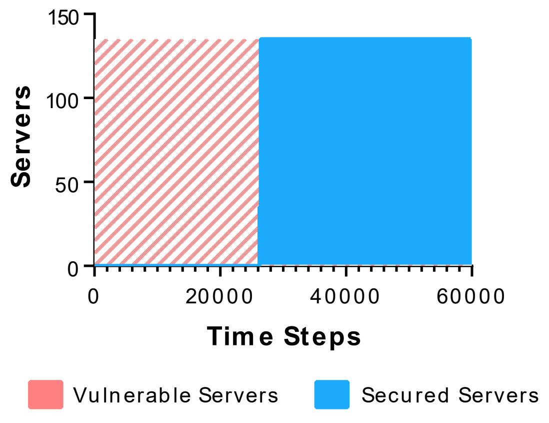
Figure 5.1 depicts the maintenance progress of each strategy in this scenario. All solutions start by updating the 35 servers initially empty. The main difference among them is the period taken by the virtual machine migrations performed after updating the empty servers. The proposed heuristic managed to update the remaining servers 3425 time steps before the second-best solution, the First-Fit heuristic. In other words, the other evaluated strategies exposed 100 servers to attacks for at least 3425 time steps longer than the proposed heuristic.



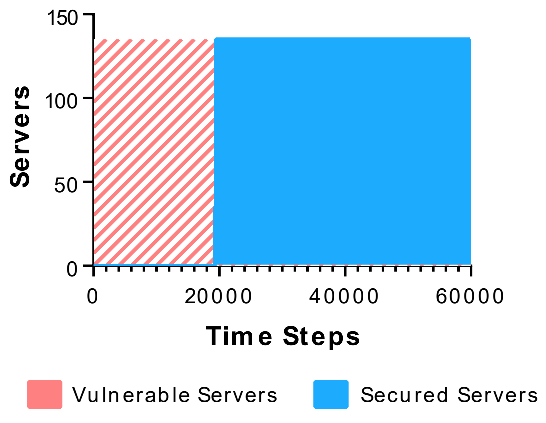
(a) Best-Fit



(b) First-Fit



(c) Worst-Fit



(d) Proposed Heuristic

Figure 5.1: Secured and vulnerable servers during server maintenance using different approaches in the low occupation scenario.

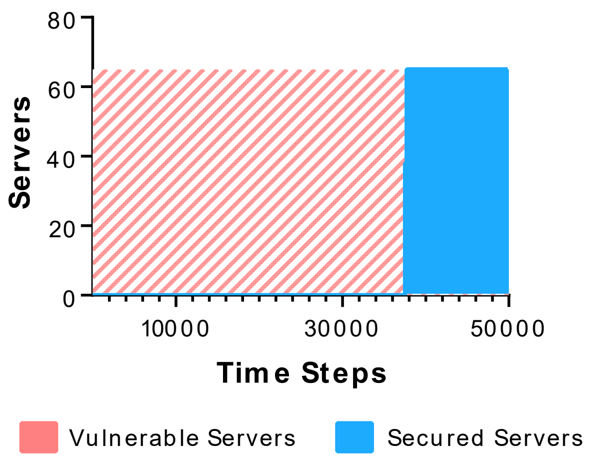
In addition to migrating virtual machines from servers that would not be possibly emptied in the current step, Best-Fit heuristic was seriously penalized by its attempt on indiscriminately consolidating virtual machines as much as possible. In the context of this study, migrating a virtual machine to a non-updated server implicates that the same virtual machine will have to be migrated at least one more time, in order to empty its host server.

Although consolidating resources leads to better use of resources, the Best-Fit heuristic considered all servers as candidates for hosting virtual machines, including the non-updated ones. As a result, 176 unnecessary migrations were performed by exchanging virtual machines among non-updated servers. This behavior not only resulted in the most extensive maintenance time but also in the biggest Vulnerability Surface, as the Best-Fit heuristic had to perform several extra migrations before being able to update all servers.

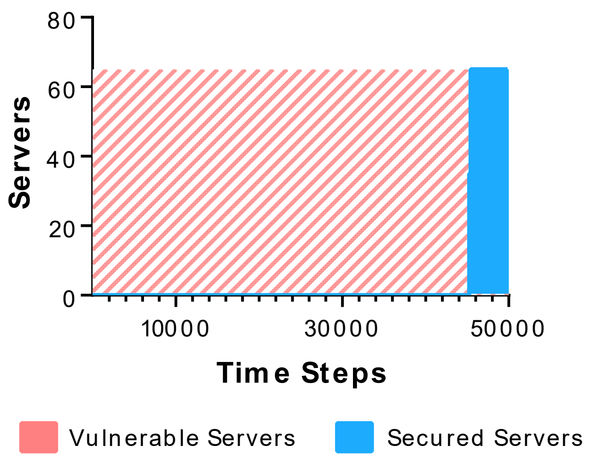
5.3 Analysis on Medium Occupation Scenario

The goal of this analysis is to investigate the behavior of different approaches in data center scenarios with a medium occupation rate (50%). In this scenario, we consider a set of 65 physical servers: 30 small, 20 medium, and 15 large.

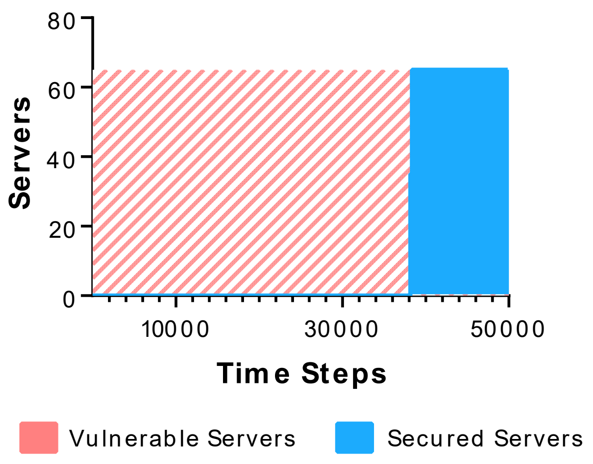
Unlike the first workload, this scenario does not include empty servers on the initial placement. Therefore, it is intended to examine the effectiveness of the evaluated solutions when it is strictly necessary to migrate virtual machines to non-updated servers.



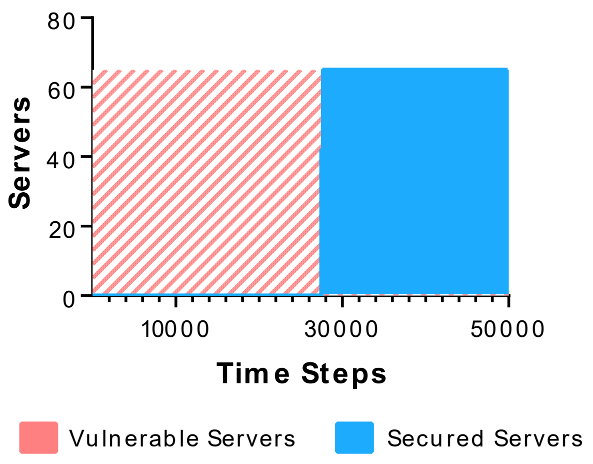
(a) Best-Fit



(b) First-Fit



(c) Worst-Fit



(d) Proposed Heuristic

Figure 5.2: Secured and vulnerable servers during server maintenance using different approaches in the medium occupation scenario.

The results obtained executing each of the observed heuristics are presented in Table 5.5. Similar to the first workload, the proposed heuristic achieves better results compared to the other solutions.

Once again, delaying virtual machine migrations that do not immediately lead to an unoccupied server and avoiding migrations to not updated servers minimized the overall maintenance time and number of migrations required by the proposed heuristic to update the data center.

Table 5.5: Experiment results for the medium occupation scenario.

Metrics	**Strategy**			
	Best-Fit	**First-Fit**	**Worst-Fit**	**Proposed Heuristic**
Maintenance Time	37696	45409	38343	27677
Virtual Machine Migrations	212	264	215	155
Vulnerability Surface	2181112	3043295	2353890	1166276

Figure 5.2 represents the maintenance progress of each of the evaluated heuristics. It is possible to observe that all heuristics take a considerable amount of time to update servers. However, by effectively choosing which virtual machines to migrate and the servers to host them, the proposed heuristic manages to update a higher number of servers in a smaller time interval.

The main advantage of the proposed heuristic over the other solutions consists of choosing to update servers with a set of smaller virtual machines or those with few virtual machines. This decision led to a shorter average time to empty servers in the first iteration. As a result, the proposed heuristic managed to update more servers within a shorter period compared to the other solutions.

In this workload, having no empty servers in the initial placement resulted in a better result for strategies that focus on consolidating virtual machines. Indeed, consolidating virtual machines was key to the positive results achieved not only by the proposed heuristic but also by the Best-Fit heuristic, which, unlike in the first workload, achieved the second-best results.

5.4 Analysis on High Occupation Scenario

The goal of this analysis is to investigate which decisions have the biggest impact on the effectiveness of maintenance strategies on data centers with high occupation rates. In this scenario, we consider a data center with 42 heterogeneous servers hosting 100 virtual machines. In this workload, the average data center's occupation is 75%.

The results obtained during the execution of the analyzed heuristics in this workload are depicted in Table 5.6. The results showed that the proposed heuristic could significantly overcome the other strategies on high occupation scenarios.

The overall higher occupation rate presented in this workload impacted on the number of iterations necessary for updating the data center on all heuristics. Figure **??** shows

Table 5.6: Experiment results for the high occupation scenario.

Metrics	**Strategy**			
	Best-Fit	**First-Fit**	**Worst-Fit**	**Proposed Heuristic**
Maintenance Time	32377	32730	28614	26471
Virtual Machine Migrations	185	188	162	135
Vulnerability Surface	1570447	1667182	1401141	841688

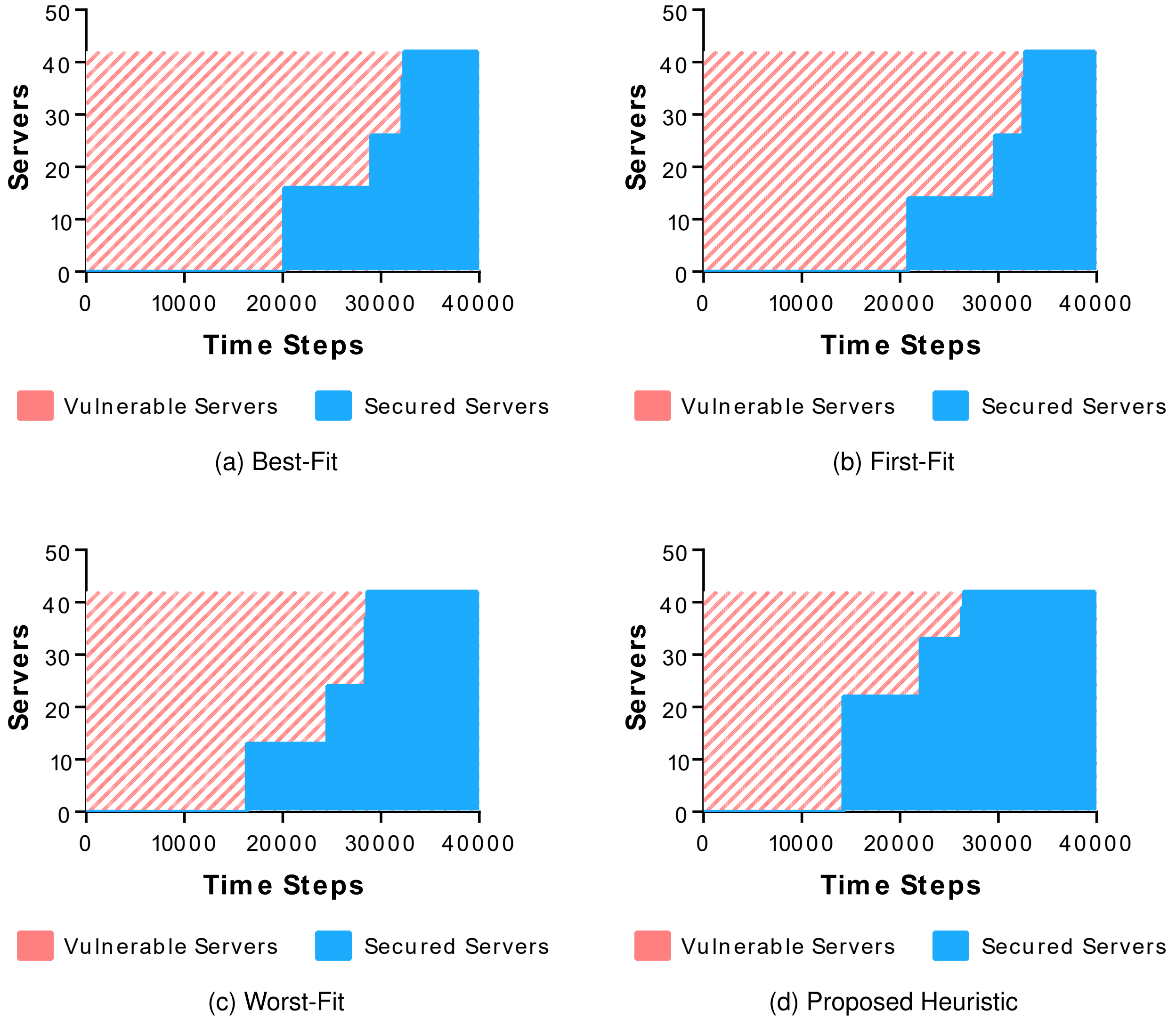


Figure 5.3: Secured and vulnerable servers during server maintenance using different approaches in the high occupation scenario.

the maintenance progress of each solution. In this scenario, heuristics were able to update fewer servers simultaneously. This situation occurred since fewer resources were available in servers so that fewer virtual machines could be migrated to empty servers.

In this scenario, all solutions presented gains regarding the Vulnerability Surface in up to 45.6%. Once again, properly choosing which virtual machine migrations to perform was key for the positive results achieved by the proposed solution. On the one hand, the

First-Fit heuristic was the one that took longer to update the data center (32730 time steps), as a result of performing 188 migrations during this process. The ineffective achievement of First-Fit contrasts to the results of the proposed heuristic, that was able to safeguard all servers in only 26471 time steps (difference of 6259 time steps), by reducing the number of migrations to 135.

5.5 Final Remarks

Based on the experiment results, it is possible to verify that the proposed heuristic effectively minimizes the time required to perform server maintenance in cloud data centers. To achieve this, the proposed heuristic tries to avoid the migration of virtual machines to not updated servers as much as possible, since each virtual machine migrated to a not updated server will have to be migrated at least one more time during the maintenance to empty its host server.

During the maintenance process, the proposed heuristic also focuses on improving the Vulnerability Surface. For this purpose, it analyzes the workload of each server and chooses for emptying the servers whose virtual machines will take less time to be migrated. Consequently, it manages to empty a subset of servers to maintenance before the other solutions, which means that these servers will be vulnerable for attacks during a shorter period.

6. CONCLUSIONS AND FUTURE WORK

In this chapter, we present the conclusions and contributions of this work. We also briefly present a list of publications achieved during the master's course period.

6.1 Conclusions

Server maintenance is a critical task when cloud data centers are exposed to attackers. In these scenarios, safeguarding as many servers as possible is crucial to minimize the damage. However, applying security patches may require restarting servers. On the one hand, operators could choose to stop all applications during maintenance. However, this decision would severely impact applications' availability requirements. On the other hand, updating servers one at a time could avoid hurting applications performance, but it would expose the data center to attacks for an unnecessary amount of time.

Previous investigations proposed different strategies for reducing server maintenance duration while keeping the maintenance as transparent as possible to applications. These studies mainly focus on finding optimal scheduling for maintenance tasks that have a predefined deadline. Their solutions concentrate on delaying or advancing maintenance in order to perform the update when few users are accessing applications. However, they do not consider critical security patching scenarios, where maintenance must be performed as soon as possible. In these scenarios, delaying maintenance even for a few moments could mean giving space for attackers to invade customers' applications.

Therefore, another approach should be considered. Evaluating new solutions based on the same traditional metrics such as maintenance time and number of virtual machine migrations would lead to solutions similar to the existing ones, that achieve good results on minimizing the overall maintenance time, but neglect the amount of time servers stay exposed to attacks.

This work presents a novel maintenance evaluation metric called Vulnerability Surface, which assesses the efficiency of operational decisions on safeguarding servers during maintenance. We first show that Vulnerability Surface is able to efficiently assess maintenance strategies even when they suggest the same number of migrations and take the same amount of time to update servers. Then, we present a heuristic algorithm that focuses on achieving a balance between reducing the time necessary for performing maintenance and minimizing the period servers remain vulnerable to attackers.

Experiments were conducted in simulated data center scenarios with different occupation rates, evaluating the proposed solution alongside Best-Fit, Worst-Fit, and First-Fit heuristics. The results indicate that the proposed solution overcomes these strategies, being able to safeguard servers sooner and to minimize the maintenance time. The proposed solution achieves better Vulnerability Surface results by giving priority to emptying servers whose virtual machines demand less time to be migrated. Regarding the maintenance time improvements, they result from the effort made by the proposed solution on avoiding migrations that would not immediately lead to a server update and on prioritizing the migration of virtual machines to updated servers.

Therefore, the contributions of this work can be summarized as follows:

- The modeling the Vulnerability Surface, a metric that aids operators to measure the efficiency of maintenance strategies on security patching scenarios on cloud data centers;
- The proposal of a heuristic algorithm that performs maintenance decisions to minimize the amount of time cloud servers remain exposed to attacks. At the same time, it reduces the amount of time necessary to perform server maintenance on cloud data centers.

6.2 Future Research Directions

In this study, we focus on providing a heuristic algorithm for updating servers that host only traditional applications (which have most of its features within a single instance, called monolith). However, several investigations show the increasing adoption of microservice-based applications for better resource usage [26] [8]. In this paradigm, components are divided into different instances and communicate with each other through message-passing mechanisms.

As part of the same application, microservices should not be deployed too far from each other due to latency issues. Therefore, when servers are being prepared to undergo maintenance, the microservices hosted on it should be re-deployed on servers close to each other to avoid communication overhead. Besides, microservices are usually hosted on containers, which present weaker isolation than virtual machines. As a result, improper co-location of containers could lead to performance interference issues [28] [12]. Accordingly, looking for efficient solutions to ensure the quality of service delivered by microservice applications during data center maintenance represents a research topic that could be addressed in future researches.

6.3 Achievements

In addition to the contributions mentioned above, we further participated in some other studies on correlated topics on virtualization and resource management on cloud and edge computing. These publications are listed next:

- **IAGREE: Infrastructure-agnostic Resilience Benchmark Tool for Cloud Native Platforms**
  Paulo Souza, Wagner Marques, Rômulo Reis, Tiago Ferreto.
  International Conference on Cloud Computing and Services Science (CLOSER), 2019.
- **Performance-Aware Energy-Efficient Processes Grouping for Embedded Platforms**
  Paulo Souza, Wagner Marques, Marcelo Conterato, Tiago Ferreto, Fábio Rossi.
  IEEE Symposium on Computers and Communications (ISCC), 2018.
- **The Impact of Parallel Programming Interfaces on the Aging of a Multicore Embedded Processor**
  Ângelo Vieira, Paulo Souza, Wagner Marques, Marcelo Conterato, Tiago Ferreto, Marcelo Luizelli, Fábio Rossi, Jorji Nonaka.
  IEEE International Symposium on Circuits and Systems (ISCAS), 2019.
- **Improving the Trade-Off between Performance and Energy Saving in Mobile Devices through a Transparent Code Offloading Technique**
  Rômulo Reis, Paulo Souza, Wagner Marques, Tiago Ferreto, Fábio Rossi.
  International Conference on Cloud Computing and Services Science (CLOSER), 2019.
- **Multilevel resource allocation for performance-aware energy-efficient cloud data centers**
  Fábio Rossi, Paulo Souza, Wagner Marques, Marcelo Conterato, Tiago Ferreto, Arthur Lorenzon, Marcelo Luizelli.
  IEEE Symposium on Computers and Communications (ISCC), 2019.
- **Towards Balancing Energy Savings and Performance for Volunteer Computing through Virtualized Approach**
  Fábio Rossi, Tiago Ferreto, Marcelo Conterato, Paulo Souza, Wagner Marques, Rodrigo Calheiros, Guilherme Rodrigues.
  International Conference on Cloud Computing and Services Science (CLOSER), 2019.
- **Evaluating container-based virtualization overhead on the general-purpose IoT platform**
  Wagner Marques, Paulo Souza, Fábio Rossi, Guilherme Rodrigues, Rodrigo Calheiros, Marcelo Conterato, Tiago Ferreto.
  IEEE Symposium on Computers and Communications (ISCC), 2018.
- **Reducing energy consumption in SDN-based data center networks through flow consolidation strategies**
  Marcelo Conterato, Tiago Ferreto, Fábio Rossi, Wagner Marques, Paulo Souza.
  ACM/SIGAPP Symposium on Applied Computing (SAC), 2018.

REFERENCES

[1] Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I.; Warfield, A. “Xen and the art of virtualization”, *ACM SIGOPS operating systems review*, vol. 37–5, October 2003, pp. 164–177.

[2] Buyya, R.; Yeo, C. S.; Venugopal, S. “Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities”. In: IEEE International Conference on High Performance Computing and Communications, 2008, pp. 5–13.

[3] Buyya, R.; Yeo, C. S.; Venugopal, S.; Broberg, J.; Brandic, I. “Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility”, *Future Generation computer systems*, vol. 25–6, June 2009, pp. 599–616.

[4] Canella, C.; Genkin, D.; Giner, L.; Gruss, D.; Lipp, M.; Minkin, M.; Moghimi, D.; Piessens, F.; Schwarz, M.; Sunar, B.; Van Bulck, J.; Yarom, Y. “Fallout: Leaking data on meltdown-resistant cpus”. In: ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 769–784.

[5] Clark, C.; Fraser, K.; Hand, S.; Hansen, J. G.; Jul, E.; Limpach, C.; Pratt, I.; Warfield, A. “Live migration of virtual machines”. In: USENIX Symposium on Networked Systems Design and Implementation, 2005, pp. 273–286.

[6] Gill, P.; Jain, N.; Nagappan, N. “Understanding network failures in data centers: measurement, analysis, and implications”, *ACM SIGCOMM Computer Communication Review*, vol. 41–4, August 2011, pp. 350–361.

[7] Gunawi, H. S.; Hao, M.; Suminto, R. O.; Laksono, A.; Satria, A. D.; Adityatama, J.; Eliazar, K. J. “Why does the cloud stop computing?: Lessons from hundreds of service outages”. In: ACM Symposium on Cloud Computing, 2016, pp. 1–16.

[8] Hasselbring, W. “Microservices for scalability: keynote talk abstract”. In: ACM/SPEC on International Conference on Performance Engineering, 2016, pp. 133–134.

[9] Hines, M. R.; Deshpande, U.; Gopalan, K. “Post-copy live migration of virtual machines”, *ACM SIGOPS operating systems review*, vol. 43–3, July 2009, pp. 14–26.

[10] Islam, M.; Razzaque, A.; Islam, J. “A genetic algorithm for virtual machine migration in heterogeneous mobile cloud computing”. In: International Conference on Networking Systems and Security, 2016, pp. 1–6.

[11] Jadeja, Y.; Modi, K. “Cloud computing-concepts, architecture and challenges”. In: International Conference on Computing, Electronics and Electrical Technologies, 2012, pp. 877–880.

[12] Jha, D. N.; Garg, S.; Jayaraman, P. P.; Buyya, R.; Li, Z.; Ranjan, R. "A holistic evaluation of docker containers for interfering microservices". In: IEEE International Conference on Services Computing, 2018, pp. 33–40.

[13] Josep, A. D.; Katz, R.; Konwinski, A.; Gunho, L.; Patterson, D.; Rabkin, A. "A view of cloud computing", *Communications of the ACM*, vol. 53–4, April 2010, pp. 50–58.

[14] Kocher, P.; Horn, J.; Fogh, A.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Mangard, S.; Prescher, T.; et al.. "Spectre attacks: Exploiting speculative execution". In: IEEE Symposium on Security and Privacy, 2019, pp. 1–19.

[15] Leiserson, C. E. "Fat-trees: universal networks for hardware-efficient supercomputing", *IEEE transactions on Computers*, vol. 100–10, October 1985, pp. 892–901.

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

[17] Lipp, M.; Schwarz, M.; Gruss, D.; Prescher, T.; Haas, W.; Fogh, A.; Horn, J.; Mangard, S.; Kocher, P.; Genkin, D.; et al.. "Meltdown: Reading kernel memory from user space". In: USENIX Security Symposium, 2018, pp. 973–990.

[18] Mell, P.; Grance, T. "The nist definition of cloud computing", *NIST Special Publication*, vol. 800, September 2011, pp. 1–3.

[19] Okuno, S.; Iikura, F.; Watanabe, Y. "Maintenance scheduling for cloud infrastructure with timing constraints of live migration". In: IEEE International Conference on Cloud Engineering, 2019, pp. 179–189.

[20] Silva Filho, M. C.; Monteiro, C. C.; Inácio, P. R.; Freire, M. M. "Approaches for optimizing virtual machine placement and migration in cloud environments: A survey", *Journal of Parallel and Distributed Computing*, vol. 111, January 2018, pp. 222–250.

[21] Slama, W. B.; Brahmi, Z.; et al.. "Interference-aware virtual machine placement in cloud computing system approach based on fuzzy formal concepts analysis". In: IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2018, pp. 48–53.

[22] Soomro, A. K.; Shaikh, M. A.; Kazi, H. "Ffd variants for virtual machine placement in cloud computing data centers", *International Journal of Advanced Computer Science and Applications*, vol. 8–10, October 2017, pp. 261–269.

[23] Tang, M.; Pan, S. "A hybrid genetic algorithm for the energy-efficient virtual machine placement problem in data centers", *Neural processing letters*, vol. 41–2, April 2015, pp. 211–221.

[24] Teng, F.; Yu, L.; Li, T.; Deng, D.; Magoulès, F. “Energy efficiency of vm consolidation in iaas clouds”, *The Journal of Supercomputing*, vol. 73–2, July 2017, pp. 782–809.

[25] van Schaik, S.; Milburn, A.; Österlund, S.; Frigo, P.; Maisuradze, G.; Razavi, K.; Bos, H.; Giuffrida, C. “RIDL: Rogue in-flight data load”. In: IEEE Symposium on Security and Privacy, 2019, pp. 88–105.

[26] Villamizar, M.; Garcés, O.; Castro, H.; Verano, M.; Salamanca, L.; Casallas, R.; Gil, S. “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud”. In: Computing Colombian Conference, 2015, pp. 583–590.

[27] Wang, X.; Chen, X.; Yuen, C.; Wu, W.; Wang, W. “To migrate or to wait: Delay-cost tradeoff for cloud data centers”. In: IEEE Global Communications Conference, 2014, pp. 2314–2319.

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

[29] Xavier, M. G.; Neves, M. V.; Rossi, F. D.; Ferreto, T. C.; Lange, T.; De Rose, C. A. “Performance evaluation of container-based virtualization for high performance computing environments”. In: Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2013, pp. 233–240.

[30] Xing, Y.; Zhan, Y. “Virtualization and cloud computing”. In: Future Wireless Networks and Information Systems, Zhang, Y. (Editor), 2012, pp. 305–312.

[31] Yanagisawa, H.; Osogami, T.; Raymond, R. “Dependable virtual machine allocation”. In: IEEE International Conference on Computer Communications, 2013, pp. 629–637.

[32] Youseff, L.; Butrico, M.; Da Silva, D. “Toward a unified ontology of cloud computing”. In: Grid Computing Environments Workshop, 2008, pp. 1–10.

[33] Yu, S.; Tian, Y.; Guo, S.; Wu, D. O. “Can we beat ddos attacks in clouds?”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 25–9, July 2013, pp. 2245–2254.

[34] Zheng, Z.; Li, M.; Xiao, X.; Wang, J. “Coordinated resource provisioning and maintenance scheduling in cloud data centers”. In: IEEE International Conference on Computer Communications, 2013, pp. 345–349.

[35] Zheng, Z.; Wang, J.; Ren, J.; Hou, W.; Wang, J. “Least maintenance batch scheduling in cloud data center networks”, *IEEE Communications Letters*, vol. 18–6, June 2014, pp. 901–904.

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