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

Paulo Silas Severo de Souza[1], **Tiago Coelho Ferreto (Advisor)**[1]

[1]Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Ipiranga Avenue, 6681 - Building 32 – Porto Alegre – RS – Brazil

Abstract. *As cyberattacks against the cloud become more frequent, operators must define efficient maintenance strategies to safeguard data centers. Existing maintenance strategies strive to minimize the maintenance duration and the number of migrations. However, such solutions overlook the period that servers wait for their update, which represents a vulnerability window that attackers can exploit. Accordingly, this study introduces a novel metric, Vulnerability Surface, which assesses maintenance strategies regarding servers' exposure. In addition, we present Salus, a heuristic that minimizes servers' exposure during maintenance. Experimental results show that Salus reduces the Vulnerability Surface by 19.44% compared to baseline strategies.*

1. Introduction

With the rise of cloud computing, customers can access computing resources from anywhere through the Internet without any commitment to physical infrastructure. In addition, Service Level Agreements (SLAs) guarantee that a minimum level of service is maintained [1]. As a result, cloud operators must regularly perform corrective maintenance to ensure cloud infrastructure can deliver the expected performance.

Unlike routine maintenance that usually does not require a strict completion deadline, safeguarding the cloud against security vulnerabilities usually means that patching must be applied promptly to mitigate the chances of attack propagation across the data center. In such a scenario, defining which servers to update and how to do so determine the effectiveness of maintenance strategies.

Previous research concentrates on minimizing the maintenance duration and number of migrations. Throughout this study, we demonstrate that such a strategy does not ensure maintenance effectiveness on safeguarding servers against threats. Therefore, we introduce the concept of Vulnerability Surface, which aids cloud operators in evaluating the effectiveness of maintenance strategies on minimizing security breaches. We also present Salus, a heuristic that reduces servers during maintenance. In summary, this work makes the following contributions:

- We introduce the Vulnerability Surface, a novel metric that effectively represents how long maintenance strategies expose servers to attackers.
- We present Salus, a heuristic algorithm that safeguard servers 19.44% faster than other solutions during data center maintenance.

The remainder of this study is organized as follows. Section 2 gives an overview of maintenance in cloud data centers. Section 3 shows how we formulate the maintenance problem. Sections 4 and 5 describe the Vulnerability Surface metric and our heuristic algorithm for performing maintenance in cloud data centers. Section 6 presents the performance evaluation we conducted to validate our proposal. Finally, Section 7 concludes the document.

2. Maintenance in Cloud Data Centers

Providing guarantees such as high performance and security constitutes the core DNA of cloud computing, wherein organizations do not have to invest in their own infrastructure [1]. In such a scenario, cloud equipment must regularly undergo maintenance to ensure the quality of service meets the high standards promised on the SLAs and applications stay safeguarded against potential threats.

Security experts fight against threats like Distributed Denial of Service (DDoS) since the 90s developing solutions to protect the cloud [2]. However, recent waves of attacks caught the security community's attention. For instance, major cloud providers like Amazon Web Services[1] and Microsoft Azure[2] reported that some of their users could experience downtime and bottleneck due to maintenance performed in haste to safeguard their infrastructure against speculative execution attacks [3] [4].

Previous research on data center maintenance concentrates on different goals. Ayoub et al. [5] present heuristics for evacuating data centers threatened by natural disasters while minimizing application downtime and network occupation. Ying et al. [6] introduce Raven, a migration scheduler that shortens the maintenance time in data centers. Wang et al. [7] propose heuristics for reducing maintenance batches necessary to update data centers under strict regulatory rules. Yazidi et al. [8] strive to minimize network saturation and QoS degradation during maintenance by considering inter-VM communication. Despite their contributions, these investigations overlook servers' exposure against cyberattacks.

3. Problem Formulation

This section describes how we model the maintenance process in cloud data centers. For convenience, Table 1 describes the symbols used henceforth. The cloud data center maintenance process is divided in a set of iterations $\mathbb{U}$. Each maintenance iteration $U_i \in \mathbb{U}$ may comprise two operations: *(i) server patching* and *(ii) VM migrations*. In such a scenario, we represent a set of n servers denoted as $\mathbb{S} \leftarrow \{S_1, S_2, ..., S_n\}$. The capacity of a server S_j is given by a vector $C_j \leftarrow (CPU, Memory, Disk)$.

The set of m VMs accommodated in the servers $\in \mathbb{S}$ is represented by $\mathbb{V} \leftarrow \{V_1, V_2, ..., V_m\}$. The demand of a VM V_k is denoted by a vector $D_k \leftarrow (CPU, Memory, Disk)$. The placement of VMs on servers at each maintenance iteration $U_i \in \mathbb{U}$ is represented by a $\mathbb{U} \times \mathbb{S} \times \mathbb{V}$ matrix $x_{i,j,k} \in \{0, 1\}$ that gets 1 if server S_j hosts VM V_k at maintenance iteration U_i and 0 otherwise.

We assume that servers hosting VMs cannot be updated. Therefore, we need to relocate all VMs from a server before it undergoes maintenance. For conciseness, from now on, we use *"drained"* to denote servers hosting no VMs.

The update status of a server $S_j \in \mathbb{S}$ at U_i is given by $w_{i,j} \in \{0, 1\}$. We assume that $\sum_{j=1}^{n} w_{1,j} = 0$ in the beginning of iteration U_1, which means that all servers must undergo maintenance to get security patches $\mathbb{P} \leftarrow \{P_1, P_2, ..., P_n\}$. Updating S_j with its patch P_j takes φ_j units of time. Once a patch P_j is applied, a set of sanity check tasks G_j is executed $\in \mathbb{G}$ to verify the system integrity. Running G_j takes $\varkappa_j$ units of time. Therefore, updating S_j takes $\varphi_j + \varkappa_j$ units of time.

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

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

Table 1. Notation used in this paper.

Symbol	Description
n	Number of servers
$\mathbb{S}$	Set of servers
C_j	Capacity vector of a server S_j
m	Number of virtual machines
$\mathbb{V}$	Set of virtual machines
D_k	Demand vector of a virtual machine V_k
$\mathbb{U}$	Set of maintenance iterations
$x_{i,j,k}$	Binary matrix that indicates whether S_j hosts V_k at iteration i or not
$w_{i,j}$	Update status of server S_j at maintenance iteration U_i
$\mathbb{P}$	Set of patches that must be applied to the servers
φ_j	Amount of time needed to apply a patch P_j
$\mathbb{G}$	Set of sanity check tests used to verify the integrity of patches $\in \mathbb{P}$
$\varkappa_j$	Amount of time needed to run a set of sanity check tests G_j
μ_k	Amount of time needed to save the state of V_k
ν_k	Amount of time needed to restore the state of V_k
ϑ	Network bandwidth available for migrations
ε_i	Duration of maintenance iteration U_i
$\partial_{i,j}$	Update cost of S_j

For relocating VMs, we consider a cold migration scheme [9]. Equation 1 denotes ϖ_k, which represents the migration time of a VM V_k. To migrate V_k, first its state is stored in the origin server. This process takes $\mu_k \in \mathbb{N}^+$ units of time. Then, the VM state is transferred throughout the network to the destination server. The duration of this process depends on the VM size (more specifically, $D_{k,Memory}$ and $D_{k,Disk}$) and the network bandwidth ϑ. Lastly, the VM state is restored in the destination server. Such a process takes $\nu_k \in \mathbb{N}^+$ units of time.

$$\varpi_k \leftarrow \mu_k \; + \; \frac{D_{k,Memory} + D_{k,Disk}}{\vartheta} \; + \nu_k \tag{1}$$

At the beginning of each maintenance iteration $U_i \in \mathbb{U}$, all drained servers simultaneously undergo maintenance to receive their patches. Then, VM migrations take place, draining servers to patch them in the next iteration, U_{i+1}. Accordingly, the duration of U_i is given by ε_i, denoted in Equation 2.

$$\varepsilon_i \leftarrow \overbrace{max\left(\{\varphi_j + \varkappa_j | j \in \mathbb{N}^+ \leqslant n \; \wedge |w_{i,j} - w_{i-1,j}| = 1\}\right)}^{\textbf{Server Patching}} + \overbrace{\sum_{j=1}^{n}\sum_{k=1}^{m} \varpi_k \cdot |x_{i-1,j,k} - \; x_{i,j,k}|}^{\textbf{VM Migrations}} \tag{2}$$

4. Vulnerability Surface Metric

Existing studies show that reducing the number of migrations and the maintenance duration somehow optimizes the maintenance process. However, these metrics overlook that some actions can be either right or wrong based on when they are taken. Consequently, these metrics do not account for inappropriate decisions such as precipitated migrations that may delay server updates.

Accordingly, we present the concept of Vulnerability Surface (VS), which computes servers' exposure during maintenance. At the end of each maintenance iteration, $U_i \in \mathbb{U}$, VS accounts for the elapsed maintenance time (given by $\sum_{q=1}^{i} \varepsilon_q$) and the number of vulnerable servers (given by $\sum_{j=1}^{n} w_{i,j}$). Therefore, the overall VS of a maintenance strategy is given by $\sum_{i=1}^{|\mathbb{U}|} \sum_{q=1}^{i} \varepsilon_q \times \sum_{j=1}^{n} w_{i,j}$.

To illustrate how VS assesses server exposure, we present a sample maintenance scenario in Figure 1. In such an example, two strategies (S1 and S2) keep servers vulnerable for different periods despite taking the same amount of time to complete the maintenance and performing the same number of migrations. Unlike the existing metrics, VS catches that difference and gives S1 a better (lower) score. For simplicity purposes, in this example, VM migrations take $10 + VM\_size$ units of time, and server updates take 60 units of time.

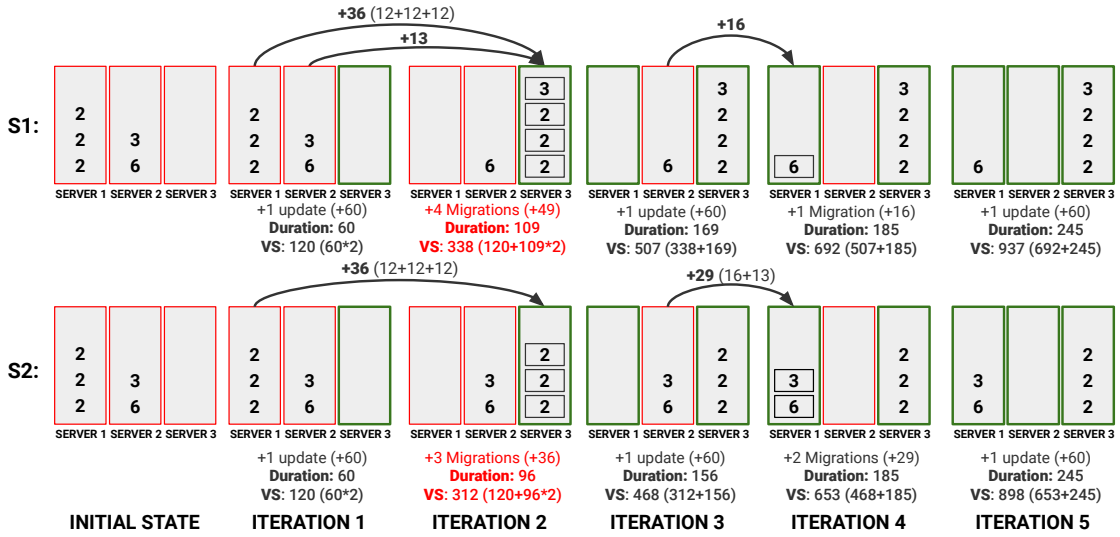


Figure 1. Sample maintenance scenario that demonstrates how the number of migrations and maintenance duration conceals ineffective decisions.

5. Proposed Heuristic for Server Maintenance

This section presents Salus, our proposed heuristic for updating cloud servers in critical security scenarios. At first, Salus patches all non-updated servers that are not hosting VMs in the data center (Algorithm 1, lines 2–5). Then, it starts migrating VMs to drain out the remaining non-updated servers (Algorithm 1, lines 6–22).

Salus decides if each server S_j is getting drained at an iteration U_i based on a cost function $\partial_{i,j}$ (denoted in Equation 6), prioritizing servers with larger capacity (Equation 3) and those it can drain out (Equation 4) and update (Equation 5) within a shorter interval. Thus, maintenance iterations get shortened, and each time step is used to drain out servers that could host more VMs in upcoming iterations.

$$\mho \leftarrow \frac{1}{\sqrt[3]{C_{j,cpu} \cdot C_{j,memory} \cdot C_{j,disk}} + 1} \tag{3}$$

$$\wp \leftarrow \sum_{k=1}^{m} \left(\mu_k + \frac{D_{k,memory} + D_{k,disk}}{\vartheta} + \nu_k \right) \cdot x_{i,j,k} \tag{4}$$

$$\hbar \leftarrow \varphi_j + \varkappa_j \tag{5}$$

$$\partial_{i,j} \leftarrow (\mho \times (\wp + \hbar))^{\frac{1}{2}} \tag{6}$$

Algorithm 1: Salus Maintenance Algorithm.

```
1  while there are servers to be updated do
2      𝔼 ← Set of servers ∈ 𝕊 ready to receive their patches
3      foreach server S_j ∈ 𝔼 do
4          Patch S_j with P_j and run sanity check G_j
5      end
6      𝕄 ← Non-updated servers sorted by ascending ∂
7      𝔸 ← {}
8      foreach server S_j ∈ 𝕄 do
9          ℍ ← VMs hosted by S_j sorted by descending demand
10         𝕎 ← 𝕊 − {𝔸 ∪ S_j}
11         if Servers ∈ 𝕎 can host all VMs hosted by S_j then
12             foreach V_k ∈ ℍ do
13                 Sort servers ∈ 𝕎 by update status and demand
14                 foreach server S_l ∈ 𝕎 do
15                     if S_l has resources to host V_k then
16                         Migrate V_k to S_l
17                         break
18                     end
19                 end
20                 𝔸 ← 𝔸 ∪ S_j
21             end
22         end
23     end
24 end
```

Before performing any migration from a given server S_j, Salus checks if the other servers can accommodate all the VMs S_j hosts. In that way, it avoids migrating VMs from servers that will not get drained in the current maintenance iteration. Consequently, Salus manages to shorten maintenance iterations, which leads to servers getting safeguarded earlier.

While migrating VMs, Salus sorts VMs in decreasing order by their demand (Algorithm 1, Line 9). Then, it populates $\mathbb{W}$ with the list of candidate hosts for each VM (Algorithm 1, Line 12) based on their update status (prioritizing updated servers) and occupation rate (servers with higher occupation get preference). Such a decision brings two benefits: (i) it avoids unnecessary migrations in the long term, as VMs migrated to non-updated servers will be migrated at least one more time during the maintenance, whenever its new host gets drained to receive the patch; (ii) it allows better packing of VMs.

6. Performance Evaluation

We evaluate Salus against three baseline heuristics named First-Fit-like, Best-Fit-like, and Worst-Fit-like in three simulated data center scenarios with different occupation rates (25%, 50%, 75%). All experiment assets can be found in a public GitHub repository[3].

The data center network comprises a Fat-Tree network topology with links containing a bandwidth of 1 Gbit/s. We do not allow simultaneous migrations to ensure the QoS for the applications not being migrated.

We consider three capacity and demand configurations for the servers and the VMs (Table 2), assined based on an uniform distribution. We define the initial VM placement using a Random-Fit heuristic. We update servers with three patch types with lengths = [300, 900, 2700]. Each patch type has a sanity check duration = [600, 1800, 5400]. We assign these patch configurations to the servers using an uniform distribution.

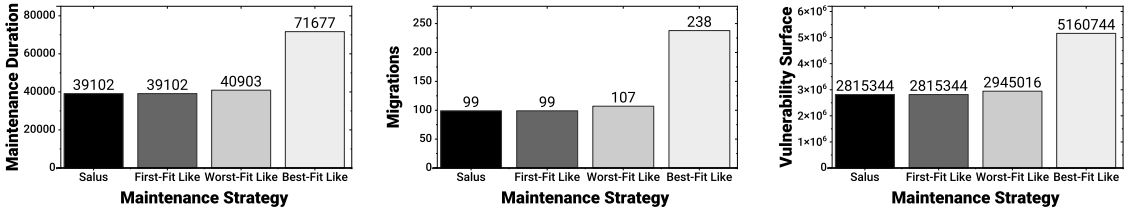
[3] `https://github.com/GRIN-PUCRS/cloud-simulator`

Table 2. Server and virtual machine configurations used during the evaluation.

Configuration	Servers			Virtual Machines		
	CPU	Memory	Disk	CPU	Memory	Disk
Small	4	4	32	1	1	8
Medium	8	8	64	2	2	16
Large	16	16	128	4	4	32

6.1. Low Occupation Scenario

Figure 2 presents the results of the second scenario. In this scenario, the decision-making necessary to safeguard servers as early as possible is quite simple, and the First-Fit-like heuristic is capable of updating all servers as fast as Salus, performing just 99 migrations (1 migration per VM). On the other hand, Best-Fit-Like and Worst-Fit-like were seriously penalized by migrating several VMs to non-updated servers.

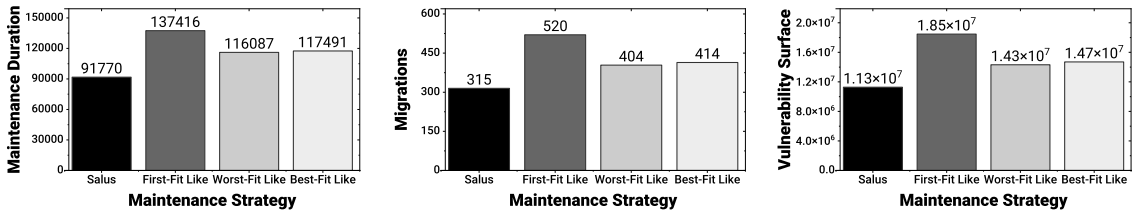


(a) Maintenance Duration **(b)** VM Migrations **(c)** Vulnerability Surface

Figure 2. Experiment results for the low occupation scenario.

6.2. Medium Occupation Scenario

Figure 3 presents the results of the second scenario. Salus achieved the best results in this scenario by draining the servers that demand shorter patch and migration time. Besides, it focuses on migrating VMs to updated servers. Consequently, Salus manages to patch 62 out of the 128 servers 40% faster than the other heuristics while performing only half of the migrations.

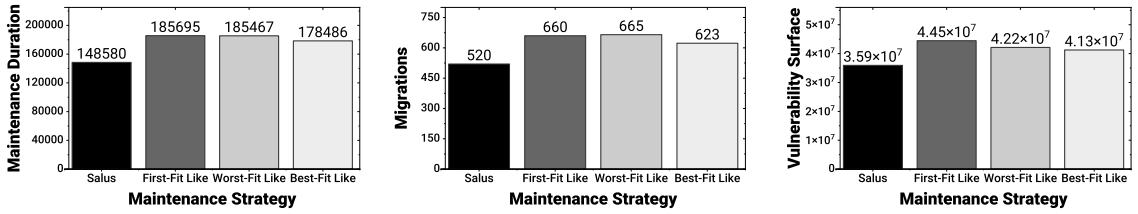


(a) Maintenance Duration **(b)** VM Migrations **(c)** Vulnerability Surface

Figure 3. Experiment results for the medium occupation scenario.

6.3. High Occupation Scenario

Figure 4 presents the results of the second scenario. Here, Salus manages to safeguard half of the data center 31% faster than the other heuristics by performing two critical decisions. Firstly, it prioritizes the servers it can drain and patch within a shorter period, shortening maintenance iterations. Secondly, it focuses on updating servers with larger capacity, as they can host more VMs in upcoming maintenance iterations after receiving their patches.



(a) Maintenance Duration **(b)** VM Migrations **(c)** Vulnerability Surface

Figure 4. Experiment results for the high occupation scenario.

7. Conclusions and Achievements

This study demonstrates that current maintenance strategies neglect the period that servers remain exposed to cyberattacks during server maintenance. Accordingly, we introduce a novel metric, called Vulnerability Surface, for assessing server exposure during data center maintenance. In addition, we present a heuristic algorithm designed to safeguard servers as early as possible during maintenance. A paper describing our contributions is under review by the Journal of Parallel and Distributed Computing. During the development of this study, we also contributed to the community with the following publications:

- **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.
- **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.
- **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.
- **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 Energy and Thermal Efficiency of Anomaly Detection Algorithms in Edge Devices**
  Felipe Rubin, Paulo Souza, Wagner Marques, Rômulo Reis, Fábio Rossi, Tiago Ferreto.
  IEEE International Conference on Information Networking (ICOIN), 2020.

Acknowledgements

This work was supported by the PDTI Program, funded by Dell Computadores do Brasil Ltda (Law 8.248 / 91). The authors acknowledge the High-Performance Computing Laboratory of the Pontifical Catholic University of Rio Grande do Sul (LAD-IDEIA/PUCRS, Brazil) for providing support and technological resources, which have contributed to the development of this project and to the results reported within this research.

References

[1] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *2008 10th IEEE International Conference on High Performance Computing and Communications*. Ieee, 2008, pp. 5–13.

[2] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[3] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium*, 2018, pp. 973–990.

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

[5] O. Ayoub, A. de Sousa, S. Mendieta, F. Musumeci, and M. Tornatore, "Online virtual machine evacuation for disaster resilience in inter-data center networks," *IEEE Transactions on Network and Service Management*, 2021.

[6] C. Ying, B. Li, X. Ke, and L. Guo, "Raven: Scheduling virtual machine migration during datacenter upgrades with reinforcement learning," *Mobile Networks and Applications*, pp. 1–12, 2020.

[7] L. Wang, H. V. Ramasamy, and R. E. Harper, "Scheduling physical machine maintenance on qualified clouds: What if migration is not allowed?" in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 485–492.

[8] A. Yazidi, F. Ung, H. Haugerud, and K. Begnum, "Affinity aware-scheduling of live migration of virtual machines under maintenance scenarios," in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2019, pp. 1–4.

[9] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, F. Xia, and S. A. Madani, "Virtual machine migration in cloud data centers: a review, taxonomy, and open research issues," *The Journal of Supercomputing*, vol. 71, no. 7, pp. 2473–2515, 2015.