Reducing Power Consumption during Server Maintenance on Edge Computing Infrastructures

Felipe Pfeifer Rubin
School of Technology - PUCRS
Porto Alegre, Brazil
felipe.rubin@edu.pucrs.br

Paulo Severo de Souza
School of Technology - PUCRS
Porto Alegre, Brazil
paulo.severo@edu.pucrs.br

Tiago Ferreto
School of Technology - PUCRS
Porto Alegre, Brazil
tiago.ferreto@pucrs.br

ABSTRACT

Edge servers must routinely undergo maintenance to ensure the environment's performance and security. During maintenance, applications hosted by outdated servers must be relocated to alternative servers to avoid downtime. In distributed edges with servers spread across large regions, ensuring that applications are not migrated to servers too far away from their users to avoid high latency hardens the maintenance planning. In addition, the limited power supply of edge sites restricts the list of suitable alternative hosts for the applications even further. Past work has focused on optimizing maintenance or increasing the power efficiency of edge computing infrastructures. However, no work addresses both objectives together. This paper presents Emma, a maintenance strategy that reduces power consumption during edge server maintenance without excessively extending maintenance time or increasing application latency. Experiments show that Emma can minimize power consumption during maintenance by up to 26.48% compared to strategies from the literature.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**;
• **Information systems** → **Computing platforms**;

KEYWORDS

Edge Computing, Maintenance, Power Consumption, Update, Infrastructure

ACM Reference Format:
Felipe Pfeifer Rubin, Paulo Severo de Souza, and Tiago Ferreto. 2023. Reducing Power Consumption during Server Maintenance on Edge Computing Infrastructures. In *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), March 27–March 31, 2023, Tallinn, Estonia*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3555776.3577739

1 INTRODUCTION

Edge computing delivers low latency by performing computations at the network edge, near data sources, rather than solely on the cloud [22]. While allocating resources nearby relieves latency issues, it also brings new challenges in power consumption, as the edge infrastructure exhibits constraints regarding computing power and network bandwidth [2, 3]. This issue gets even more complicated as infrastructure operators conduct maintenance.

Maintenance imposes performance and energy impacts on edge infrastructures, as servers remain unavailable during patching but still consume energy. Applications must be relocated to other servers to avoid downtime. At the same time, they must remain close enough to users to deliver the expected performance levels [22]. Recent studies employed different strategies to reduce the power consumption of the edge infrastructure [4, 27, 28] or proposed maintenance strategies to update cloud and edge servers [18, 23, 24, 31]. However, none of these works aimed at reducing the power consumption of the edge infrastructure during maintenance.

This paper presents Emma, a maintenance strategy that reduces the power consumed by the edge infrastructure when updating edge servers while also relocating applications during the process to avoid downtime and satisfy latency constraints concerning users' locations. To the best of our knowledge, this is the first work to optimize power consumption during maintenance on edge infrastructures.

The remaining of this work is organized as follows. Section 2 discusses the fundamental concepts involving power management and maintenance planning at the edge. Section 3 reviews related studies and highlight our contributions to the literature. Sections 4 and 5 introduce the system model and our proposed edge server maintenance strategy. Section 6 details the experiments used to evaluate our approach. Finally, Section 7 concludes the paper.

2 BACKGROUND

This section discusses the fundamental concepts related to our study, including resource management in edge computing infrastructure toward power consumption reductions and seamless maintenance execution.

2.1 Edge Computing

Edge computing [21] arises with the promise of delivering low latency by bringing computational power to the edge of the network so that applications can be executed closer to users' devices [22]. In a typical edge infrastructure, edge servers are interconnected by switches and hosting users' applications while users' devices connect to access points to communicate with their applications [10].

While the proximity between users and their applications provides an inherent advantage over cloud data centers in terms of latency, it also raises new concerns for managing infrastructure resources due to the limited processing power of edge servers and the bandwidth capacity of the edge network [14]. From a network perspective, ensuring the same robustness of cloud data centers is practically unfeasible due to the geographical dispersion of infrastructure resources [2]. At the same time, such dispersion is necessary for hosting applications nearby users' locations.

2.2 Power Consumption

In the past few decades, power conservation in cloud data centers became a topic of interest for reducing costs and managing system performance [17]. Conveniently, the shift of some applications to the edge reduced the power consumption of network devices and servers in cloud data centers. However, by promoting latency-sensitive applications to run closer to users, managing the power consumption levels of the edge infrastructure became increasingly difficult. Accommodating applications with near real-time requirements on the edge infrastructure demand a considerable amount of power from edge servers [10]. Thus, it becomes necessary to explore different approaches that can reduce the power consumption of the edge infrastructure.

At a hardware level, reducing power consumption of idle resources is possible with low-power sleep modes at the cost of requiring some time to revert to a normal state(*power-saving states* [16]), adjustable component operational frequency at the cost of reducing processing capacity (*Dynamic Voltage and Frequency Scaling (DVFS)* [9]), and the eventual power off at the cost of an increase in boot time. Power savings are also achievable using software-based approaches employed on cloud data centers, such as migration of applications from overloaded servers consuming too much power [13] or consolidating applications from underloaded servers and temporarily disabling idle ones [1] at the cost of impacting the quality of service.

2.3 Maintenance

The European Standard *BS EN 13306:2017* [25] defines maintenance as a "combination of all technical, administrative and managerial actions during the life cycle of an item intended to retain it in, or restore it to, a state in which it can perform the required function.".

Maintenance is a delicate process as it is essential for the infrastructure. Lack of proper maintenance accounts for over a third to half of downtime events and over 30% to 40% of outages originating from hardware failures [30]. However, maintenance also places a heavy burden on the edge infrastructure. Relocating applications is a common prerequisite to conducting maintenance operations on servers.

As servers become unavailable during their update, not all may be updated simultaneously. Whether by reprovisioning stateless applications or migrating stateful applications (maintaining runtime information) [20], relocating applications is necessary to avoid downtime. Operating on idle and offline servers prevents modifications from causing errors and impacting the applications' performance [15]. At the same time, with the increase in network usage on relocating applications and requiring more servers in use while others update, an increase in power consumption is also expected.

3 RELATED WORK

This section discusses existing research related to our study. First, Section 3.1 reviews research efforts on power management in edge infrastructures. Then, Section 3.2 discusses related efforts in maintenance planning. Finally, Section 3.3 highlights our contributions to the state of the art.

3.1 Power Efficiency

The resource-constrained nature of the edge highlights the importance of efficient and sustainable use of resources. Most existing power-conserving efforts exploit hardware features such as sleep states [19, 28] and DVFS [12, 27] to reduce the power consumption of underutilized servers. Whereas the first approach temporarily shuts down inactive servers, the latter dynamically adjusts the operating frequency of the hardware components while servers remain active. Other strategies include exploiting virtualization technology to relocate applications to servers with lower power consumption and shut down idle hosts [4].

3.2 Maintenance

As maintenance typically stresses the infrastructure, early efforts focused on reducing maintenance time through server prioritization policies [18, 23, 31]. However, such approaches focused on cloud data centers, relocating applications to servers distant from users. In response to such limitations, more recent strategies started including user locality in migration decisions to make maintenance less intrusive for latency-sensitive applications [24].

3.3 Our Contributions

As discussed in Sections 3.1 and 3.2, previous research efforts have tackled the topics of power saving and maintenance planning separately. This work fills this gap with a novel maintenance strategy that reduces the infrastructure's power consumption during edge server updates while taking care of the quality of service of edge applications.

4 SYSTEM MODEL

This section presents our system model. We first describe infrastructure assets that host users' applications, and later we detail the maintenance process to update edge servers. We represent the map as a set of hexagonal cells based on the model presented by Aral et al. [3]. The coordinates of any location on the map correspond to the whole area of a cell, such that there is no location where cells intersect.

The edge infrastructure comprises a set of network devices $\mathcal{N}$, located on each map cell, interconnected by a set of network links $\mathcal{L}$. A network device is an abstract representation of a co-located network switch and base station. Network switches form the cabled network infrastructure, and base stations act as wireless access points for a set of users $\mathcal{U}$ that access a set of applications $\mathcal{A}$. Applications are hosted by a set of heterogeneous edge servers $\mathcal{S}$ that is also part of the edge infrastructure. Each network device's coverage area (base station) corresponds to a cell, and any user inside the cell connects to it. Each edge server has an exclusive location and is directly connected to the network device in its cell.

We consider the update of all servers by applying a set of patches, one at each host, and a set of sanity checks after each patch. We assume that servers must reboot for patches to take effect. We also assume that the applications hosted on the servers are stateful and, therefore, relocating applications is only possible with migrations. Every edge server must be drained by migrating its hosted applications to other servers to avoid downtime. Accordingly, the maintenance continues over several steps, representing the elapsed time, until all edge servers are updated.

A network device is modeled as $\mathcal{N}_j = (w_j)$, where w_j represents the wireless latency of $\mathcal{N}_j$'s base station. A network link is represented as $\mathcal{L}_u = (l_u, b_u, z_u)$, where l_u, b_u, and z_u denote the link's latency, bandwidth capacity, and power consumption when active, respectively. We model an edge server as $\mathcal{S}_i = (c_i, d_i, \phi_i, \psi_i, p_i)$, where c_i and d_i represent the server's resource capacity and demand, respectively, ϕ_i denotes the server's outdated status (1 when $\mathcal{S}_i$ is outdated and 0 otherwise), ψ_i denotes the time needed to update $\mathcal{S}_i$, and p_i is the $\mathcal{S}_i$'s power consumption.

We assume that an edge server continues to consume power even when idle (not hosting any application) since, without any intervention, it will remain powered on. We assume that with direct intervention, servers can be powered off and powered on at any time as long as they are idle. The power consumed may differ for each server, but we consider it a constant part of the maximum power consumed when the server operates at full load. When hosting applications, in addition to the power consumed for being powered on, the server consumption grows according to its utilization (applications' demand over the server's capacity).

An application is represented as $\mathcal{A}_k = (g_k, q_k)$, where g_k denotes $\mathcal{A}_k$'s resource demand and q_k denotes the maximum latency tolerated by the $\mathcal{A}_k$'s user $\mathcal{U}_k$. We denote the hosting relationship of an application $\mathcal{A}_k$ on an edge server $\mathcal{S}_i$ with $h_{k,i} = 1$ if it is hosted on the server and 0 otherwise. We assume that it is possible to estimate the power consumption impact that could result from hosting the application $\mathcal{A}_k$ on the edge server $\mathcal{S}_i$, denoted as $x_{k,i}$. We assume that there is a communication path $R_{k,i}$ between an application $\mathcal{A}_k$, hosted on the edge server $\mathcal{S}_i$ and its user $\mathcal{U}_k$, which is computed with Dijkstra's shortest path algorithm [8] using link latency l_u as the weight. The accumulated latency on a communication path $R_{k,i}$ is denoted as $\theta_{k,i}$ and given by Eq. 1, where w_j is the wireless delay of the network device to which the user $\mathcal{U}_k$ is connected.

$$\theta_{k,i} = w_j + \sum_{u \in R_{k,i}} l_u \tag{1}$$

A Service Level Agreement (SLA) violation occurs whenever the accumulated latency of the application's communication path exceeds the application's latency threshold (i.e., $\theta_{k,i} > q_k$). We assume that during a migration, the accumulated latency remains the same as before the migration started. Therefore, if the latency SLA of an application is in a state of violation before the start of the migration, it will continue to be the same during the migration. Only after the migration ends and the communication path changes to the new host of the application the accumulated latency is recomputed for the new communication path.

Migrating an application implies transferring its data from one server to another using the network infrastructure. We assume that the data to transfer for the application $\mathcal{A}_k$ depends on its disk demand g_k, which includes the necessary space to store the information of the application state. We denote the migration relationship of an application $\mathcal{A}_k$ using the link $\mathcal{L}_u$ as $m_{k,u}$. The migration path comprising all links used to migrate $\mathcal{A}_k$ is computed using Dijkstra's shortest path algorithm [8], where the transfer weight of a link is denoted by τ_u and calculated according to Eq. 2.

$$\tau_u = \frac{max(1, \sum_{k=1}^{\mathcal{A}} d_k * m_{k,u})}{b_u} \tag{2}$$

We limit the scope of migrations to originate from outdated servers during draining. Also, after a migration begins, it cannot be canceled, nor can the migration path be changed. However, there is no limit to the number of applications that can be migrated at any given moment other than the limited resources available on edge servers. In case a network link $\mathcal{L}_u$ is included in more than one migration path simultaneously, the Max-Min Fairness algorithm [6] is used to determine the bandwidth shares that are allocated to each application migration.

5 PROPOSED STRATEGY

This section presents Emma, our strategy that reduces the power consumption during edge server maintenance while preserving the quality of service of applications.

Algorithm 1: Emma

```
1  $\mathcal{S} \leftarrow$ List of edge servers
2  $outdated \leftarrow \{\}$
3  foreach $\mathcal{S}_i \in \mathcal{S}$ do
4      if $\phi_i$ then
5          if $idle(\mathcal{S}_i)$ then
6              $update(\mathcal{S}_i)$
7          else
8              $outdated \leftarrow outdated \cup \mathcal{S}_i$
9      else
10         if $idle(\mathcal{S}_i)$ then
11             $poweroff(\mathcal{S}_i)$
12 Sort "outdated" by $\delta$ (asc.), max power as tiebreaker (desc.)
13 foreach $\mathcal{S}_i \in outdated$ do
14     $\mathcal{S}' \leftarrow \mathcal{S} - \mathcal{S}_i - \{$outdated being drained$\}$
15     $\mathcal{A}' \leftarrow$ List of applications on $\mathcal{S}_i$
16     Sort $\mathcal{A}'$ by demand (desc.)
17     if Servers $\mathcal{S}'$ have capacity to host applications $\mathcal{A}'$ then
18         foreach $\mathcal{A}_k \in \mathcal{A}'$ do
19             Sort $\mathcal{S}'$ by $\sigma$ (asc.)
20             foreach $\mathcal{S}'_i \in \mathcal{S}'$ do
21                 if $c_i - d_i \geq d_k$ then
22                     if $\mathcal{S}'_i$ is powered off then
23                         $poweron(\mathcal{S}'_i)$
24                     Compute migration path with $\lambda$
25                     Migrate $\mathcal{A}_k$ to $\mathcal{S}'_i$
26                     Mark $\mathcal{S}_i$ as being drained
27                     break
```

5.1 Power Off Idle Server After Update

We make power-efficient decisions and opportunistically power off edge servers once they are updated. Emma starts by observing the state of every server. Idle servers, those not hosting applications, can be updated immediately (Alg. 1, lines 5–6). Soon after a server is updated, an opportunity presents itself where an updated server is idle and, therefore, can be powered off to save power (Alg. 1, lines 10–11). Outdated servers hosting applications must be drained before they can be updated (Alg. 1, lines 7–8).

5.2 Order to Drain Servers

The draining process of outdated servers hosting applications conducted by Emma (Alg. 1, line 12) organizes servers in ascending order of drain weight δ, (Eq. 3). The drain weight of a server is calculated as the sum of its inverse normalized capacity, normalized demand, and normalized update duration. Min-Max Normalization [11] is used to operate values of different scales. The geometric mean is used to express the values of capacity and demand. Update duration includes the duration of the patch and the sanity checks. The maximum power consumption in descending order acts as a tiebreaker for servers with the same weight.

$$\delta = 1 - norm(c_i) + norm(d_i) + norm(\psi_i) \quad (3)$$

The earlier servers with more capacity are available, the sooner they can host applications. The less demand applications have on a server, fewer data must be migrated, and sooner draining completes. The faster the estimated update of a server is, the sooner it can host applications or be powered off. Even if servers have a similar capacity, demand, and update duration, the power they consume can differ. Using the maximum power as a tiebreaker causes more power-hungry servers to be drained, updated, and powered off earlier, saving power.

5.3 Selection of Migration Candidates

When selecting candidate servers to host the applications it needs to migrate, Emma excludes the server being drained itself and outdated servers that have already progressed in draining (Alg. 1, line 14). This distinction leaves as remaining candidates, servers already updated, whichever their power state is, and outdated servers that have not started evacuating their applications. Servers that are being updated cannot host applications and therefore are excluded.

The evacuation of applications follows a descending order of most resource-demanding first (Alg. 1, lines 15–16). The more demanding an application is, the more the power consumed for hosting is expected. While the power consumed by an application depends on the server and the application's demand for each type of resource, an approximate value for demand can be obtained from a geometric mean. The limited resources available for applications become more apparent once servers are updated simultaneously. To avoid prematurely draining servers, a preliminary verification (Alg. 1, line 17) is put in place to decide whether the current resources available on the candidate servers are sufficient to drain the outdated server completely.

Selecting the most appropriate candidates for migrations depends on the servers and each application's user. Emma organizes the candidate servers (Alg. 1, line 19) in ascending order of candidate weight, denoted as σ. The candidate weight for hosting $\mathcal{A}_k$ on $\mathcal{S}_i$ is calculated by Eq. 4 as the sum of the normalized expected network communication latency between the candidate and the application's user, the normalized power consumption impact of hosting the application, and a bit representation of whether the server is outdated or not (1 if outdated, 0 otherwise). The power consumption impact from hosting an application also accounts for the candidate's idle power if it is currently powered off.

$$\sigma = norm(\theta_{k,i}) + norm(\chi_{k,i}) + \phi_i \quad (4)$$

Emma prioritizes migrating applications to updated nearby hosts displaying lower power consumption impact by hosting the applications. Migrating an application to an outdated server implies having to migrate such an application at least one more time to complete the maintenance, as its new host will eventually need to be drained and updated. Accordingly, prioritizing migrations to already updated servers avoids unnecessary migrations and reduces the overall maintenance time. In addition, we strive to select the updated servers with a lower power consumption profile, as once applications are relocated to updated servers, they remain in such hosts until the end of maintenance.

Emma assumes that applications are positioned nearby users before the maintenance. As such, migrating them to nearby servers will likely require few links while keeping applications with sufficiently low latency. Once migrations use few links, the chances of dividing the bandwidth are low—which helps reduce the migration duration—while reducing the network power consumption. After sorting the edge servers, Emma migrates the application to the best candidate server found with enough free resources (Alg. 1, lines 21–27). If the target server is currently powered off, it is powered on before initiating the migration (Alg. 1, lines 22–23).

5.4 Selection of Links for Migration

Since migrations also impact the power utilization of the network infrastructure, Emma computes the links to use in the migration path (Alg. 1, line 24) using Dijkstra's shortest path weighting network links (edges) with a migration weight, denoted as λ. Since the links forming the path are selected greedily, instead of using normalization, the migration weight is calculated by Eq. 5 as the default weight attributed to the link plus a β factor.

$$\lambda = \tau_u + \beta \tag{5}$$

Regarding τ_u, which is the default weight attributed to links in our system model when computing the migration path, we represent the β factor in this work as the demand of the application that is computing its migration path, doubled in case the link is not currently in use, all divided by the bandwidth of the link. In conformance with our system model, we represent the alternative migration weight in Eq. 6 as the sum of the migration demand for migrating applications using the link plus the β factor.

$$\lambda = \frac{\sum_{k=1}^{\mathcal{A}} d_k * m_{k,u} + d_k * (2 - min(1, \sum_{k=1}^{\mathcal{A}} m_{k,u}))}{b_u} \tag{6}$$

Migrations using links with more bandwidth can transfer applications faster. Even if the associated switch ports must be turned on, the faster the migration completes, the earlier the ports can be powered off. To reduce the number of switch ports in use only for migrations (not for the communication of users and applications), Emma selects links already in use. However, the more simultaneous migrations share a link's bandwidth, the slower applications transfer and the longer the ports associated with the links have to remain powered on. Hence, Emma selects links already in use with bandwidth shared by the least number of migrations.

After computing the migration path and starting the migration (Alg. 1, lines 24–25), the outdated server initially hosting the application is marked (Alg. 1, line 26) as being drained to avoid new incoming migrations from prolonging the maintenance duration and the draining continues with the next application.

6 PERFORMANCE EVALUATION

This section presents the experiments performed to evaluate Emma's effectiveness in edge server maintenance scenarios. First, we describe our methodology. Then, we discuss the performance of compared strategies regarding maintenance metrics and power consumption.

6.1 Experiments Description

We consider an edge infrastructure with 100 heterogeneous edge servers with linear power model [5] interconnected by 400 network devices forming a partially-connected mesh topology spread across a 20x20 hexagonal map grid. Edge servers are placed in random positions on the map, as shown in Figure 1. Table 1 details the edge server specifications. The network topology comprises links with 100 Mbps and 1000 Mbps distributed uniformly. Whereas 100 Mbps switch ports incur in an average latency of 4 ms and consume 0.3W, 1000 Mbps switch ports incur an average latency of 2 ms and consume 1W [7]. Unused switch ports are automatically powered off as in Conterato et al. [7]. Nonetheless, network switches are always on regardless of the occupation of network ports.

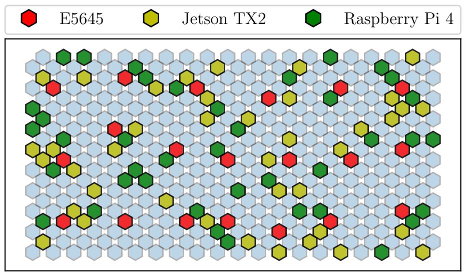

Figure 1: Edge server positioning on the map.

Table 1: Edge server specifications.

Device Model	Power Consumption (W)	CPU / RAM / Disk
Raspberry Pi 4 [26]	Idle: 2.56. Max: 7.3	4 Cores / 8 GB / 8 GB
Jetson TX2 [26]	Idle: 7.5. Max: 15	4 Cores / 8 GB / 8 GB
E5645 [29]	Idle: 63.1. Max: 200	12 Cores / 16 GB / 64 GB

In our scenario, all edge servers within the infrastructure must be updated. Each server's update process involves applying a patch and two accompanying sanity checks for validation. The duration of each patch and sanity check is established from the uniform distribution of {250, 350} and {300, 400}, respectively, as in Souza et al. [24]. Edge servers host a set of applications created with real delay SLA requirements specified by 3GPP (specifications in Table 2). To understand Emma's effectiveness in different scenarios, we create three datasets varying the number of applications to generate three datasets with the average occupation of edge servers in terms of CPU utilization in 25%, 50%, and 75%. We refer to these occupation scenarios as A (low occupation), B (medium occupation), and C (high occupation). After positioning users in random positions on the map, we define the initial application placement by hosting applications on random edge servers that are preferably close enough to the users to respect the delay SLAs.

Table 2: Types of applications used in the evaluation.

Type	Description	SLA	Demand
GPS	Indoor positioning	15 ms	CPU: 1. RAM: 1 GB. Disk: 256 MB
Gaming	Augmented reality	20 ms	CPU: 2. RAM: 2 GB. Disk: 512 MB
V2V	Vehicle-to-vehicle	30 ms	CPU: 3. RAM: 3 GB. Disk: 1024 MB

We conducted the experiments using EdgeSimPy[1], an agent-based edge computing simulator that incorporates fine-grained modeling of several resource management processes in edge infrastructures, allowing the prototyping of various resource allocation policies such as placement, migration, and maintenance. During the evaluation, we compared Emma against three literature maintenance strategies (GLB [31], Salus [23], and Lamp [24]).

The rest of this section evaluates the compared strategies in the considered scenarios regarding maintenance duration, number of migrations, number of latency SLA violations, and power consumption. As edge servers and users are positioned randomly, the presented results are the average of ten executions with different seed values in each scenario.

6.2 Maintenance Duration

Figure 2 presents the average maintenance duration results in each scenario. The major contributor to the different maintenance duration results is the number of servers that can be simultaneously updated, which is related to the infrastructure occupation. The more occupied the scenario is, the more burdensome it becomes to drain servers. Not only are more applications to evacuate, but fewer servers have enough free resources to be candidates for the destinations of migrations. Hence, fewer servers can update simultaneously, leading to a prolonged maintenance duration.

The evaluated strategies present similar results in scenarios A and C. While a low occupation allows all the compared

[1]http://edgesimpy.github.io/

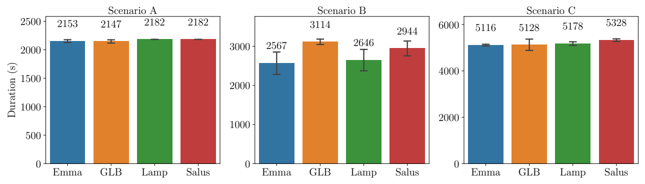

Figure 2: Average maintenance duration.

strategies to update many servers at the beginning of the maintenance, a high occupation restricts the decision space, concealing potential differences between the approaches. The most significant differences appear in scenario B. Overall, Emma and Lamp achieve the best results, draining less occupied servers first to ensure that more updated resources are available sooner. As for GLB, its consolidation approach causes many applications to be relocated to outdated servers, increasing the number of migrations and, consequently, the maintenance duration.

6.3 Number of Migrations

As we must drain edge servers before updating them, all applications must be migrated at least once, regardless of the maintenance strategy. Accordingly, our analysis focuses on migrations targeting outdated servers. The notoriety of migrations to outdated servers comes from the fact that those servers can only be temporary hosts for the application. Since they have to be drained eventually, any application migrated to them will have to be relocated at least one more time later on. All of these additional migrations can end up prolonging the maintenance duration. Figure 3 presents the number of migrations to outdated servers required by each strategy in each scenario.

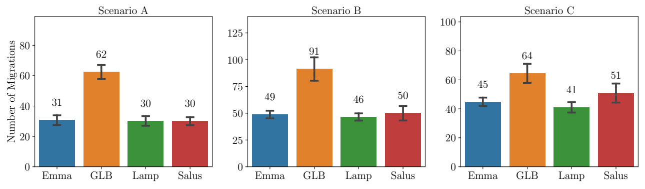

Figure 3: Number of Migrations to Outdated Servers.

GLB has a significantly higher evacuation of applications at the maintenance beginning than the other strategies. This is caused by draining more occupied servers first and aggravated by consolidating applications that we can only migrate to outdated servers that will later have to be drained. Salus causes comparatively more migrations than Emma and Lamp in scenarios B and C because it does not consider the application demand on the servers it selects to drain. This causes more applications to be consolidated on outdated servers at the beginning of the maintenance due to migration decisions. However, by draining servers with more capacity and faster updates first, fewer servers have to evacuate their applications at the beginning of the maintenance as compared to GLB.

Lamp considers the demand of the applications on servers, their capacities, and update durations when selecting which server to drain first. This causes Lamp to evacuate fewer applications than Salus at the beginning of the maintenance. Emma has an additional tiebreak rule compared to Lamp to drain more power-consuming servers first. Moreover, in addition to avoiding the violation of latency SLAs on migration decisions, Emma also tries to host applications on servers that will have the least power consumption impact.

Since Emma powers off servers right after they complete their update and the power consumption impact accounts for the static power when the server is powered off, when considering that less power-consuming servers will be drained and updated later, we can conclude that some migrations will continue to target outdated servers as the maintenance progresses. The additional migrations are acceptable for Emma as a trade-off for reducing power consumption while the maintenance progresses.

6.4 Latency SLA Violations

With the limited resources from servers nearby users, latency SLA violations are bound to happen. Even if they lead to SLA violations, some migrations are essential for the progress in draining servers and eventually completing the maintenance. Figure 4 depicts the average number of SLA violations caused by each maintenance strategy on each occupation scenario. The number of SLA violations accounts for all applications with violated SLAs in each simulation step; hence, even if the same application has its SLA violation continuously over several simulations steps, the number of violations will be the same as the number of steps (i.e., one violation per step).

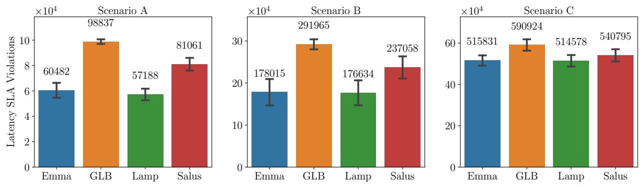

Figure 4: Latency SLA Violations.

Lamp caused the least SLA violations, and Emma came closely after. While both strategies have a similar approach in migrating applications to servers that result in the least communication latency, Emma has to balance avoiding SLA violations with reducing power consumption. Some of Emma’s decisions to migrate applications to servers with the least power impact at the cost of increased latency will lead to SLA violations which Lamp could avoid.

The other two strategies, Salus and GLB, caused more violations. While Emma and Lamp were designed with the constraints of the edge in mind, Salus and GLB were not. What led Salus to cause fewer SLA violations than GLB comes down to the number of migrations to outdated servers. Since both strategies consolidate applications without regard to the communication latency, the more migrations consolidate applications farther away, the higher the number of SLA violations will be.

6.5 Power Consumption

In scenario A, the infrastructure’s low occupation favored Emma’s power-efficient decisions. With many resources available, Emma could leave servers with higher power consumption powered off (i.e., E5645) and drain the less consuming ones (i.e., Raspberry Pi 4 and Jetson TX2), which could eventually host the applications for the rest of the maintenance. Figure 5 shows the average power consumption of the edge servers.

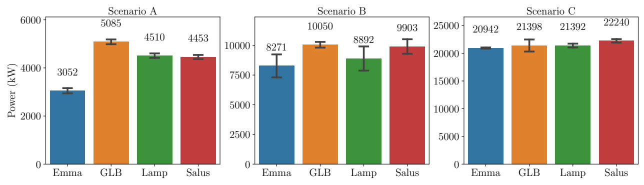

Figure 5: Power consumption of edge servers.

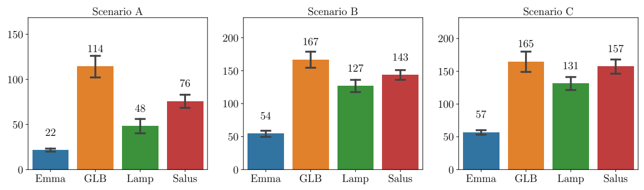

Figure 6: Power consumption incurred by migrations.

In scenarios B and C, Emma had to deal with a more occupied infrastructure, which limited the power efficiency optimization space. However, in addition to prioritizing the draining of edge servers with lower power consumption, Emma focused on using active switch ports with lower power consumption for network communication, reducing the power consumption incurred by migrations by 35.37% on average in the three scenarios, as shown in Figure 6. Lamp obtained the second best results in terms of power consumption reduction also for optimizing migrations. Although Lamp does not consider infrastructure power consumption, it conducted migrations using a smaller number of links to avoid increases in application latency, which also reduced the network power consumption.

7 CONCLUSION AND FUTURE WORK

This work presents Emma, a maintenance strategy that balances the trade-off between avoiding SLA violations and reducing the power consumed while conducting maintenance of the Edge infrastructure. Our strategy enables the update of edge servers while considering users' locations which is the deciding factor for the performance of latency-sensitive applications we explore in this work.

Simulations based on real edge server and application specifications comparing Emma against three approaches from the literature show that Emma can reduce the edge infrastructure's power consumption during maintenance by up to 26.48% without sacrificing the quality of service delivered to applications. In future work, we intend to leverage other power-saving techniques (e.g., DVFS and sleep states) to achieve higher power savings during maintenance in scenarios with mobile users and composite applications.

ACKNOWLEDGMENTS

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 for providing resources for this project.

REFERENCES

[1] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. 2015. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications* 52 (2015), 11–25.

[2] Atakan Aral and Ivona Brandić. 2020. Learning Spatiotemporal Failure Dependencies for Resilient Edge Computing Services. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2020), 1578–1590.

[3] Atakan Aral, Vincenzo Demaio, and Ivona Brandic. 2021. ARES: Reliable and Sustainable Edge Provisioning for Wireless Sensor Networks, In IEEE Transactions on Sustainable Computing. *IEEE Transactions on Sustainable Computing*, 1–12.

[4] Marios Avgeris, Dimitrios Spatharakis, Dimitrios Dechouniotis, Aris Leivadeas, Vasileios Karyotis, and Symeon Papavassiliou. 2022. ENERDGE: Distributed energy-aware resource allocation at the edge. *Sensors* 22, 2 (2022), 660.

[5] Anton Beloglazov and Rajkumar Buyya. 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience* 24, 13 (2012), 1397–1420.

[6] Dimitri P Bertsekas and Robert G Gallager. 1992. *Data networks* (2 ed.). Prentice Hall. 493–536 pages.

[7] Marcelo da Silva Conterato, Tiago Coelho Ferreto, Fábio Rossi, Wagner dos Santos Marques, and Paulo Silas Severo de Souza. 2019. Reducing energy consumption in SDN-based data center networks through flow consolidation strategies. In *Symposium on Applied Computing*. 1384–1391.

[8] Edsger W Dijkstra et al. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[9] Stijn Eyerman and Lieven Eeckhout. 2011. Fine-grained DVFS using on-chip regulators. *ACM Transactions on Architecture and Code Optimization* 8, 1 (2011), 1–24.

[10] Mohammad Goudarzi, Huaming Wu, Marimuthu Palaniswami, and Rajkumar Buyya. 2020. An application placement technique for concurrent IoT applications in edge and fog computing environments. *IEEE Transactions on Mobile Computing* 20, 4 (2020), 1298–1311.

[11] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. Data Mining: Concepts and Techniques Third Edition [M]. *The Morgan Kaufmann Series in Data Management Systems* 5, 4 (2011), 83–124.

[12] Hui Huang, Qiang Ye, and Hongwei Du. 2020. Reinforcement learning based offloading for realtime applications in mobile edge computing. In *International Conference on Communications*. IEEE, 1–6.

[13] Yeonwoo Jeong, Esrat Maria, and Sungyong Park. 2021. Towards energy-efficient service scheduling in federated edge clouds. In *International Conference on Cluster Computing*. Springer, 1–13.

[14] Congfeng Jiang, Tiantian Fan, Honghao Gao, Weisong Shi, Liangkai Liu, Christophe Cerin, and Jian Wan. 2020. Energy aware edge computing: A survey. *Computer Communications* 151 (2020), 556–580.

[15] Vincent Kherbache, Eric Madelaine, and Fabien Hermenier. 2014. Planning live-migrations to prepare servers for maintenance. In *European Conference on Parallel Processing*. Springer, 498–507.

[16] Yongpeng Liu, Hong Zhu, Kai Lu, and Xiaoping Wang. 2012. Self-adaptive management of the sleep depths of idle nodes in large scale systems to balance between energy consumption and response times. In *International Conference on Cloud Computing Technology and Science*. IEEE, 633–639.

[17] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V Vasilakos. 2014. Cloud computing: Survey on energy efficiency. *Comput. Surveys* 47, 2 (2014), 1–36.

[18] Shingo Okuno, Fumi Iikura, and Yukihiro Watanabe. 2019. Maintenance scheduling for cloud infrastructure with timing constraints of live migration. In *International Conference on Cloud Engineering*. IEEE, 179–189.

[19] Dk Siti Nur Khadhijah Pg. Ali Kumar, SH Shah Newaz, Fatin Hamadah Rahman, Gyu Myoung Lee, Gour Karmakar, and Thien-Wan Au. 2022. Green Demand Aware Fog Computing: A Prediction-Based Dynamic Resource Provisioning Approach. *Electronics* 11, 4 (2022), 608.

[20] Zeineb Rejiba, Xavier Masip-Bruin, and Eva Marín-Tordera. 2019. A survey on mobility-induced service migration in the fog, edge, and related computing paradigms. *Comput. Surveys* 52, 5 (2019), 1–33.

[21] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* 8, 4 (2009), 14–23.

[22] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[23] Paulo Souza and Tiago Ferreto. 2020. *A heuristic algorithm for minimizing server maintenance time and vulnerability surface on data centers*. Master's thesis. Graduate Program in Computer Science.

[24] Paulo S. Souza, Tiago C. Ferreto, Fábio D. Rossi, and Rodrigo N. Calheiros. 2022. Location-Aware Maintenance Strategies for Edge Computing Infrastructures. *IEEE Communications Letters* 26, 4 (2022), 848–852.

[25] British Standards Institution Staff. 2018. *BS EN 13306:2017 Maintenance. Maintenance terminology*. Standard. British Standards Institution.

[26] Ahmet Ali Süzen, Burhan Duman, and Betül Şen. 2020. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *International Congress on Human-Computer Interaction, Optimization and Robotic Applications*. IEEE, 1–5.

[27] Asfa Toor, Saif ul Islam, Nimra Sohail, Adnan Akhunzada, Jalil Boudjadar, Hasan Ali Khattak, Ikram Ud Din, and Joel JPC Rodrigues. 2019. Energy and performance aware fog computing: A case of DVFS and green renewable energy. *Future Generation Computer Systems* 101 (2019), 1112–1121.

[28] Yalan Wu, Jigang Wu, Long Chen, Jiaquan Yan, and Yuchong Luo. 2020. Efficient task scheduling for servers with dynamic states in vehicular edge computing. *Computer Communications* 150 (2020), 245–253.

[29] Muhammad Zakarya, Lee Gillam, Hashim Ali, Izaz Ur Rahman, Khaled Salah, Rahim Khan, Omer Rana, and Rajkumar Buyya. 2022. epcAware: A Game-Based, Energy, Performance and Cost-Efficient Resource Management Technique for Multi-Access Edge Computing. *IEEE Transactions on Services Computing* 15, 3 (2022), 1634–1648.

[30] Zeyu Zheng, Minming Li, Xun Xiao, and Jianping Wang. 2013. Coordinated resource provisioning and maintenance scheduling in cloud data centers. In *International Conference on Computer Communications*. IEEE, 345–349.

[31] Zeyu Zheng, Jinfan Wang, Jing Ren, Weigang Hou, and Jianping Wang. 2014. Least maintenance batch scheduling in cloud data center networks. *IEEE Communications Letters* 18, 6 (2014), 901–904.