Maintenance Operations on Cloud, Edge, and IoT Environments: Taxonomy, Survey, and Research Challenges

PAULO S. SOUZA, Federal University of Pampa, Brazil
TIAGO C. FERRETO, Pontifical Catholic University of Rio Grande do Sul, Brazil
RODRIGO N. CALHEIROS, Western Sydney University, Australia

The emergence of the Internet of Things (IoT) introduced new classes of applications whose latency and bandwidth requirements could not be satisfied by the traditional Cloud Computing model. Consequently, the IT community promoted the cooperation of two paradigms, Cloud Computing and Edge Computing, combining large-scale computing power and real-time processing capabilities. A significant management challenge in such complex infrastructure concerns the development of efficient maintenance strategies to preserve the environment's performance and security. While the abundant resources from the academic literature could support the design of novel maintenance solutions, extracting actionable insights from the existing approaches is challenging, given the massive number of published papers. Furthermore, existing review papers, which could help summarize the state-of-the-art, scope their investigations to the maintenance of certain components in particular scenarios. This work fills this gap with a broader literature analysis that covers maintenance strategies targeting physical and logical components in cloud, edge, and IoT environments. First, we introduce a taxonomy that organizes existing solutions according to several characteristics. Then, we review the literature following the taxonomy structure to facilitate the understanding of the research landscape and the comparison between existing works. Finally, we shed light on open challenges that represent promising research directions.

CCS Concepts: • **General and reference → Surveys and overviews**.

Additional Key Words and Phrases: Maintenance, Cloud Computing, Edge Computing, Internet of Things

ACM Reference Format:
Paulo S. Souza, Tiago C. Ferreto, and Rodrigo N. Calheiros. XXXX. Maintenance Operations on Cloud, Edge, and IoT Environments: Taxonomy, Survey, and Research Challenges. *ACM Comput. Surv.* XX, XX, Article XXX (XX XXXX), 36 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Cloud Computing [1] has been acknowledged as the reference model for deploying applications through the Internet since the release of popular services managed by major providers back in 2006 [2]. Whereas the cloud's domain was unchallenged for many years, advances in hardware production and telecommunications enabled the emergence of the Internet of Things [3], which introduced new classes of sensor-rich applications whose latency and bandwidth requirements could not be satisfied solely by cloud data centers due to their distance to data sources. As a response to the challenge of handling the demands of real-time IoT applications, a new computing paradigm called Edge Computing [4] was introduced as an extension of the cloud that brings computing resources

Authors' addresses: Paulo S. Souza, Federal University of Pampa, Tiaraju Avenue, 810 Ibirapuitã, Alegrete, Brazil, paulosilas@unipampa.edu.br; Tiago C. Ferreto, Pontifical Catholic University of Rio Grande do Sul, Ipiranga Avenue, 6681 Partenon, Porto Alegre, Brazil, tiago.ferreto@pucrs.br; Rodrigo N. Calheiros, Western Sydney University, Parramatta, NSW 2150, Australia, Sydney, Australia, R.Calheiros@westernsydney.edu.au.

0360-0300/XXXX/XX-ARTXXX $15.00
https://doi.org/XXXXXXX.XXXXXXX

closer to data sources to alleviate the demand upon the Internet's core and to cope with the application's low latency requirements.

Despite the contrasting characteristics between cloud, edge, and IoT infrastructures, these environments share some commonalities, including the need for meticulous maintenance strategies, which act as countermeasures against various undesired events that can affect the applications' performance and security (e.g., failures and cyber-attacks) [5] [6] [7] [8]. In this context, the wide range of tasks encompassed by maintenance work raises significant concerns about the potential damage caused by poor provisioning decisions during such activities. From a networking perspective, the increased traffic incurred by maintenance-related operations (e.g., patch distributions and application migrations) can quickly saturate the network and cause several side effects, such as packet losses and increased latency. From a computing perspective, maintenance work can affect the infrastructure stability by either making components temporarily unavailable (e.g., in cases when patches require device reboots to take effect) or by excessively increasing the concurrent demand under physical resources (e.g., when multiple applications are stacked on a single server while other servers are updated).

The potential side effects of maintenance on the infrastructure put significant pressure on IT personnel, which must have a deep understanding of maintenance-related activities to avoid the undesirable effects of such a resource-intensive activity. Several research efforts have addressed maintenance-related challenges in the Cloud-Edge-IoT ecosystem. However, summarizing and extracting insightful and actionable information from their findings and drawing parallels between multiple works is challenging due to the large number of papers in the literature. In this context, review papers stand out as valuable sources of systematic knowledge. Overall, review papers can be helpful for the community as their content can benefit both active members in the field with overviews of the state-of-the-art and indications of research subjects requiring further attention and newcomers with a rich source of learning material.

Previous review papers have provided interesting analyses of the maintenance literature in cloud, edge, and IoT environments (see Section 3.1 for details). Nonetheless, they focused on surveying the existing maintenance research efforts through specific facets (e.g., scoping the study to particular components and maintenance approaches). In addition, none provided a unified investigation exploring the commonalities and potential shareable insights from maintenance operations in these different paradigms. To the best of our knowledge, this is the first review to provide a unified analysis and categorization of the existing approaches designed to address the maintenance needs of physical and logical components in cloud, edge, and IoT environments.

In summary, the main contributions of this work are:

- We present a comprehensive literature review of the academic solutions for addressing the various maintenance needs of the physical and logical components (e.g., replacement of defective equipment and software updates) that compose cloud, edge, and IoT environments.
- We present a novel taxonomy that organizes existing maintenance solutions according to diverse characteristics (e.g., target components, maintenance approaches, and target metrics) to reduce the barrier to entry for new researchers in the field.
- We shed light on several research challenges and opportunities that represent promising directions for future investigations.

The remaining of this paper is organized as follows. Section 2 presents a background on Cloud Computing, Edge Computing, Internet of Things, and maintenance. Section 3 describes the research methodology employed during our review. Section 4 presents the proposed taxonomy and the state-of-the-art survey. Section 5 sheds light on challenges and opportunities that represent promising directions for future investigations. Finally, Section 6 concludes the paper.

2 BACKGROUND

This section presents the fundamental concepts covered in this work. First, we detail the different computing paradigms intersected by our review (i.e., Cloud Computing, Edge Computing, and the Internet of Things). Then, we discuss maintenance approaches and their role in the addressed scenarios.

2.1 Cloud Computing

Cloud Computing has been established over the years as a *de facto* standard for hosting applications on the Internet [2]. According to the *National Institute of Standards and Technology (NIST)*, the cloud's basic idea is to support users with ubiquitous, convenient, and on-demand access to a shared pool of configurable computing resources (infrastructure, platforms, and software applications) [1]. One of the key enablers for the cloud's on-demand subscription model is elasticity, which dynamically fits resources to cope with the demand [1]. When the demand rises, elastic cloud systems add resources to services, allowing them to scale according to the workload. Cloud elasticity is generally enabled by virtualization, which decouples software applications from the underlying hardware [9]. In addition to enabling on-the-fly application scaling, virtualization technologies such as Virtual Machines (VMs) and containers allow a single physical instance to host multiple applications simultaneously, improving overall resource usage.

As cloud technology matures and new players enter the market, providers define Service Level Agreements (SLAs) with increasingly strict performance promises to stand out against the competition. While ever-increasing performance expectations raise customer satisfaction, they also put pressure on IT operators, who need to create efficient maintenance strategies to allow data center resources to be repaired and updated to ensure continued compliance with performance and security requirements while causing the least possible service disruption.

2.2 Internet of Things

Despite significant advances in Information and Communications Technology, most of the interaction between physical and electronic components has been dependent on human intervention, which restricts the potential of technology to the time, attention, and accuracy constraints of human beings. To avoid this bottleneck, several discussions have pointed to the potential of the Internet of Things [3], which incorporates networking capabilities (Internet) to physical objects (Things), allowing such devices to automatically collect, communicate, and process environmental data supporting data-driven, intelligent decisions.

By encompassing a variety of devices, IoT enables several use cases, from agriculture, where smart irrigation reduces water waste and enhances harvest efficiency, to personal healthcare, where body sensors in patients provide real-time insights to doctors [10]. While the abundance of data fuels the rich use cases of IoT, it also poses a significant challenge for data processing. As IoT devices are resource-constrained, they must offload data to third-party entities, which perform the processing and send the feedback to IoT devices for on-site decision-making. In addition to making IoT communication costly, this raises concerns about potential leaks of sensitive user information such as geographical location and medical diagnoses [11]. This broad attack surface forces IoT operations teams to constantly run after efficient maintenance plans to update devices, safeguarding them from security vulnerabilities.

2.3 Edge Computing

Early IoT applications offloaded processing to the cloud, where resource-abundant data centers could easily hold the demand [12]. However, the increasing need for high bandwidth and low latency highlighted the negative consequences of the cloud's centralized model, where computing resources are distant from data sources. While the many hops necessary for communication incur network round-trip times that conflict with the real-time latency requirements of mobile applications, holding IoT processing in consolidated data centers also leads to high ingress bandwidth demand, reducing the overall network performance. Consequently, the decision to process data from such applications in the cloud became more arguable, paving the way for emerging paradigms such as Edge Computing [4]. Edge Computing refers to placing networked computing devices at the Internet's edge, close to end devices. The fundamental idea of decentralizing computing resources to the edge solves two major problems that challenge the cloud model. While distributing edge servers into different locations avoids bottlenecks at specific hotspots, the network proximity to data sources grants faster application response times.

Edge Computing deployments typically span networked compute nodes dispersed across the environment, as installing large-scale edge data centers is often unfeasible in various scenarios, such as urban centers. While the physical dispersion allows edge servers to be located only a few hops from end devices, it also introduces significant technical challenges related to edge IT operations. Once edge servers are deployed outdoors in small-sized facilities, they are inherently exposed to hardware issues (e.g., accelerated aging due to increased temperatures, power outages, physical damage, etc.) and security threats (e.g., network attacks, physical tampering, etc.). In this context, maintenance strategies are critical in ensuring that performance and security issues do not nullify the potential gains of edge infrastructure's network proximity to data sources.

2.4 Maintenance

The IT market is amidst a significant shift, with cloud and edge platforms working together to produce actionable insights from the overwhelming amount of data produced in real-time by mobile and IoT applications. To accomplish such a goal, IT operations teams managing such a three-tiered computing model (Cloud-Edge-IoT) must overcome several operational challenges, such as equipment failures, cyber-attacks, and hardware and software aging. In this context, proper maintenance planning (what to do) and scheduling (when to do it) are critical to maintaining the various software stacks and their underlying infrastructure running smoothly.

Multiple maintenance policies have been proposed over the years. Initially, enterprises invested in corrective maintenance, following a "run-to-failure" strategy where components stay in operation until they fail. Corrective maintenance relies on the assumption that maintenance cost savings are higher than the cost of disruption. However, such an assumption cannot hold in cases where high availability is required. In addition, certain assets present detectable states of degradation (e.g., failing more often with increasing age), enabling less intrusive maintenance decisions than run to failure. Figure 1 illustrates a P-F Curve [13], a typical cumulative damage model that represents the progression of misbehavior symptoms towards functional asset failure. In such scenarios, preventive measures can be taken within the so-called P-F Interval, a period between the beginning of misbehavior and the asset crash.

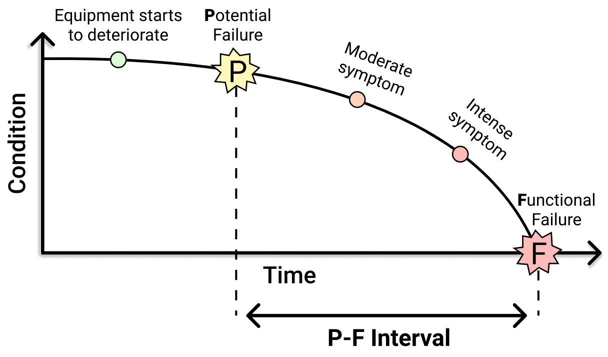

Fig. 1. Typical representation of the P-F Curve, which denotes an asset's degrading behavior towards functional failure.

The simplest preventive maintenance approach is Schedule-Based Maintenance (SBM) [14], in which assets are repaired at predefined intervals. Given that excessively long maintenance intervals increase the risk of asset failure, SBM is usually performed within significantly short periods—even if this is not always necessary—to avoid functional failures. While SBM displays a natural advantage over corrective maintenance by avoiding service disruption, it assumes that degradation progresses steadily so that failures do not occur between maintenance actions. In contrast to SBM, Condition-Based Maintenance (CBM) [14] follows a more flexible approach, triggering maintenance actions only when the asset's state has deteriorated to a certain threshold, optimistically avoiding both delayed and hasty maintenance decisions.

Preventive maintenance assumes that the degradation leaves clues for long enough for maintenance measures to be taken, which does not apply to situations where failures display non-linear progression or occur due to external factors such as cascading errors. Given such limitations, significant efforts have been made toward replacing threshold-based proactive maintenance decisions with predictive approaches [15]. Predictive maintenance techniques determine the maintenance schedule by estimating an asset's Remaining Useful Life (RUL) based on its internal attributes and the state of neighboring components. Predictive maintenance techniques are divided into two categories: model-based and data-based methodologies. While the former relies on mathematical models built through the knowledge of engineers, the latter uses statistical and machine learning algorithms to predict the asset's state through historical data [16]. Figure 2 presents a taxonomy based on Silvestri et al. [17] and Kim et al. [16] that categorizes the different maintenance approaches.

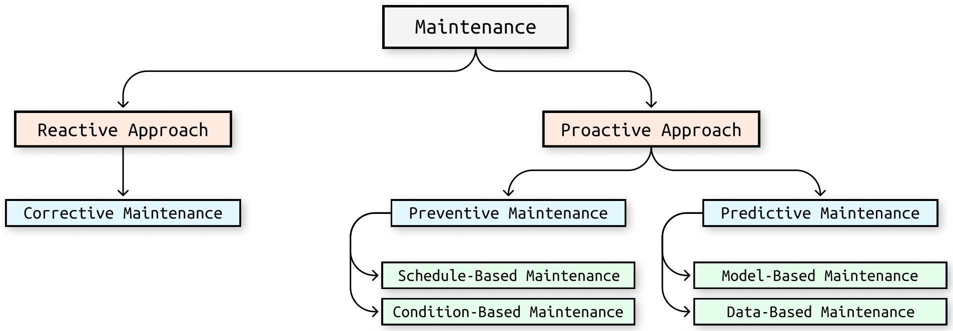

Fig. 2. Taxonomy of maintenance approaches [16, 17].

Conducting maintenance operations in complex computing environments (as is the case of cloud, edge, and IoT infrastructures) requires several decisions to be made to ensure efficiency, minimal disruption, and system integrity. During hardware component maintenance, critical decisions often revolve around whether or not to deactivate the affected systems (also known as hot swap and cold swap approaches, respectively) and whether to conduct repair procedures in the system's operational environment or move them to specialized facilities. Whereas performing maintenance without shutting down affected systems is ideal to avoid downtime, it is subject to hardware compatibility constraints. Similarly, on-site repairs offer convenience in terms of logistics but may not be feasible for more complex repair tasks.

During the maintenance of logical components, critical decisions often include deciding whether the process will be in-place or out-of-place. In-place upgrades involve applying modifications directly to the current components. Although in-place upgrades have minimal space requirements as they overwrite elements of existing components, they carry the inherent risk of system corruption if the update process encounters issues. In addition, reverting the system states can be challenging since the original data is overwritten during the update procedure. In out-of-place upgrades, new separate versions of the components are deployed and installed before replacing the old instances. While out-of-place upgrades allow for easier fallback than in-place upgrades, they require more space and resources, as they entail temporarily maintaining two versions of the components.

Maintenance operators must also determine the order in which components undergo maintenance. In this stage, maintenance can be performed all at once, potentially causing components to go through a no-service period, or it can progress gradually, which may be less disruptive but requires more sophisticated planning. Common maintenance scheduling approaches include Blue-Green Deployment and Canary Deployment.

In Blue-Green deployments, the update is done using two identical computing environments: one with the stable version (Blue) and the other with the new release (Green). Initially, users only access the Blue environment (stable version). Meanwhile, the Green environment is updated and tested. If the tests are successful, user traffic is gradually redirected to the Green environment. Figure 3 depicts a general Blue-Green deployment procedure. While Blue-Green Deployment is an effective solution for ensuring that update side effects can be isolated and mitigated without affecting the entire environment, it can be computationally expensive, as both Blue and Green environments remain operational simultaneously for a certain period.

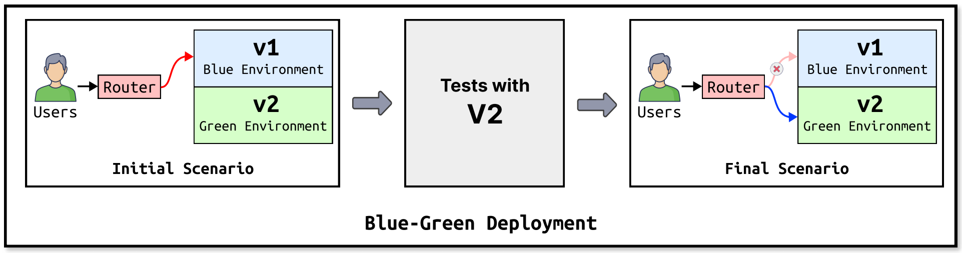

Fig. 3. Sample workflow of the Blue-Green deployment approach.

In Canary deployments (also known as rolling upgrades), components are updated gradually, avoiding compromising the entire environment in case of failures during the migration to the latest version. Figure 4 illustrates a Canary deployment procedure. While Canary deployment eliminates the need for two separate environments for maintenance (as with Blue-Green Deployment), it also requires a set of complex decisions to be made. First, it is necessary to define how many components will be updated simultaneously, which can affect the total maintenance time and compromise the infrastructure's stability in case of failure. Second, maintenance operators must define the order in which components will be updated, possibly based on multiple criteria defined according to characteristics of applications (e.g., SLAs and computational demand) and underlying infrastructure (e.g., allocation cost and power consumption).

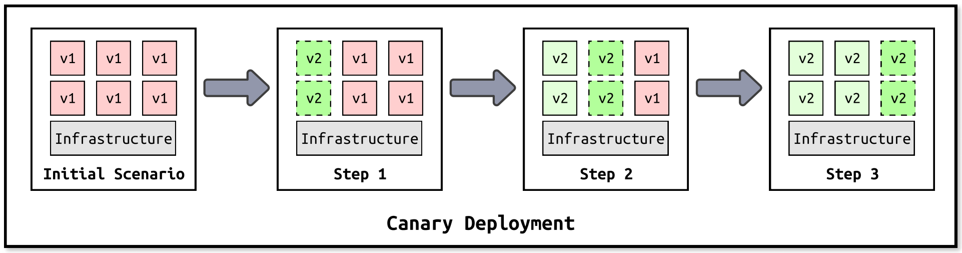

Fig. 4. Sample workflow of the Canary deployment approach.

Despite the strategic value of maintenance in preserving the continuous operation of complex ecosystems such as cloud, edge, and IoT infrastructures, there is a lack of review studies in the field, which raises the barrier to entry for new researchers. This paper fills this gap with a literature analysis and a novel taxonomy that identifies and characterizes the main research efforts in the field published between 2017 and 2023. In addition, we enumerate several open challenges, providing insights for future research endeavors. The following sections detail our methodology and literature analysis.

3 RESEARCH METHODOLOGY

This section details the research methodology. First, we provide an overview of related reviews, highlighting how our work complements their efforts (§ 3.1). After motivating our work, we present our research questions and the employed search string (§3.2). Finally, we describe our filtering protocol and an overview of the selected papers grouped by year and target paradigm (§3.3).

3.1 Related Surveys

Given the importance of maintenance operations, previous review papers have provided various analyses of the maintenance literature in cloud, edge, and IoT infrastructures. This section discusses existing literature reviews in the field and delineates how our study diverges from and builds upon these existing works, offering a more comprehensive overview of maintenance strategies in the analyzed paradigms.

Benestad et al. [18] presented a literature review on software maintenance, focusing on analyzing the objectives and attributes that drive maintenance work targeting software systems throughout their life cycles. Among the identified gaps, the authors highlighted the need for software maintenance strategies based on theoretical models that could be adapted to different contexts to complement existing approaches, which rely on empirical observations that result in a lack of extensibility.

Monperrus [19] studied the academic literature on automatic software repair, which comes into the scene helping to fix the problem without human intervention in a context where more bugs are produced than developers can fix using manual approaches. The author splits his analysis into behavioral repair and state repair. While behavioral repair consists of changing undesirable software behaviors through modifications in the source code or binaries, state repair involves fixing software issues by altering the application's state through approaches like reinitialization and checkpoint–rollback. In addition to discussing the existing automatic repair approaches found in literature, the author also provided an overview of other related software maintenance techniques. Given the ever-increasing relevance of automatic software repair, Petke et al. [20] have also surveyed the academic efforts in that field. However, instead of providing a general literature analysis, Petke et al. [20] focused on a branch of automatic software maintenance called Genetic Improvement, where automatic search methods are used to improve existing software. Among their findings, the authors highlighted the predominance of Evolutionary Algorithms for fixing bugs, improving parallelism levels, and reducing application power consumption.

Abadi et al. [21] addressed maintenance work under a different facet, with a literature review on maintenance in cloud data center facilities. The authors focused on operations tasks such as managing cooling and power supply systems, indicating open challenges related to infrastructure management activities (e.g., developing availability benchmarks for data center facilities). Some reviews have also focused on maintenance research applied to emerging subjects such as Industry 4.0 [22], which envisions a digital transformation in the manufacturing industry through IoT-related technologies. Silvestri et al. [17] surveyed the existing maintenance solutions for Industry 4.0, highlighting the need for new diagnostic and prognostic algorithms to embed intelligence into manufacturing processes. Zonta et al. [23] presented a literature review on predictive maintenance applied to Industry 4.0, introducing a taxonomy that organizes research works and describing several research opportunities, such as employing image analysis to assess the manufacturing equipment states and trigger maintenance work.

Table 1 compares our work against related reviews. Our comparison analyzes the scope of the studies in terms of addressed paradigms (i.e., cloud, edge, and IoT) and maintained components. Specifically, we categorize components into two groups: physical components (i.e., servers, network devices, and storage appliances) and logical components (ranging from low-level software like firmware and hypervisors to general-purpose applications). It is worth noting that the reviews by Benestad et al. [18], Monperrus [19], and Petke et al. [20] do not address maintenance issues specific to any particular paradigm. In summary, our review provides a broader analysis that enables a more holistic understanding of the field and the identification of potential shareable insights from maintenance operations in the different analyzed paradigms.

Table 1. Summary and comparison of the scope of this work and previous reviews.

Study	Year	Target Components		Covered Paradigms		
		Physical	Logical	Cloud Computing	Edge Computing	Internet of Things
Benestad et al. [18]	2009	×	✓	N/A	N/A	N/A
Petke et al. [20]	2017	×	✓	N/A	N/A	N/A
Monperrus [19]	2018	×	✓	N/A	N/A	N/A
Abadi et al. [21]	2020	✓	×	✓	×	×
Silvestri et al. [17]	2020	✓	✓	×	×	✓
Zonta et al. [23]	2020	✓	✓	×	×	✓
This Review	2023	✓	✓	✓	✓	✓

3.2 Research Questions and Search String

We employ three research questions (RQs), as presented in Table 2. With RQ1, answered in Section 3.3, we seek to understand the pace at which the academic community's interest in the addressed research topics has evolved in recent years. As for RQ2, the goal is to identify relevant characteristics of the selected studies. RQ2 is answered in Section 4, which comprises a taxonomy that organizes the selected research according to its characteristics. Finally, RQ3 aims to identify research opportunities. RQ3 is answered in Section 5, highlighting the open challenges within the addressed research subject.

Table 2. List of research questions addressed in this review.

Identifier	Research Question
RQ1	How has the research interest in maintenance in the fields of Cloud Computing, Edge Computing, and Internet of Things evolved between 2017 and 2023?
RQ2	What are the metrics of interest, strategies, and validation methodologies used to perform maintenance in the evaluated scenarios?
RQ3	What are the challenges and open questions on maintenance in Cloud Computing, Edge Computing, and the Internet of Things?

This review surveys academic studies indexed in three search engines: Association for Computing Machinery (ACM) Digital Library[1], Institute of Electrical and Electronics Engineers (IEEE) Xplore[2], and Scopus[3]. The search string used to retrieve research works in the selected databases is shown in Figure 5. We configured the selected databases to look for the search string in their indexed papers' titles, abstracts, and keywords.

[1]https://dl.acm.org/
[2]https://ieeexplore.ieee.org/Xplore/home.jsp
[3]https://www.scopus.com/search/form.uri

```
("Maintenance" OR "Patch" OR "Repair" OR "Update" OR "Upgrade" OR "Rejuvenation"
OR "Recovery") AND ("Cloud Computing" OR "Edge Computing" OR "Internet of Things")
```

Fig. 5. Search string used in the review.

3.3 Filtering Protocol

After collecting the returned papers, we applied a set of filtering criteria to select research papers published in academic journals and conferences containing direct contributions to the field. Papers targeting Industry 4.0 were also excluded as previous reviews already surveyed such a group of studies (see Section 3.1). The search string returned 17564 entries. After collecting the initial sample, we started filtering papers according to the methodology depicted in Figure 6. After the filtering stage, we obtained the final review list with 49 papers.

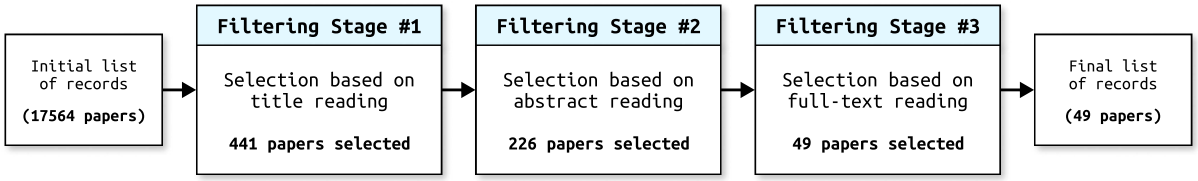

Fig. 6. Methodology used to filter papers during the review.

Figure 7 answers RQ1 by presenting the distribution of selected papers by target paradigm and year of publication. Despite the growing interest in the subject since 2019, most research efforts have focused on addressing maintenance challenges in cloud and IoT environments, highlighting the initial stage of maintenance research in edge environments. A detailed discussion on the existing contributions is presented in Section 4.

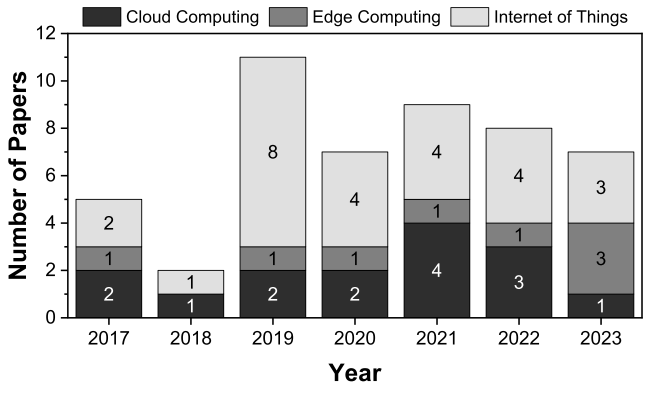

Fig. 7. Number of selected papers per year and target paradigm.

4 TAXONOMY AND SURVEY

Maintenance on Cloud-Edge-IoT ecosystems imposes significant pressure on the infrastructure, as it involves activities such as downloading patches and relocating applications from components that need to be restarted during the update. Consequently, it encompasses various resource management decisions that can significantly affect the end-user's quality of service if not appropriately planned. This section presents a systematic review of existing maintenance research in the fields of cloud, edge, and IoT. As maintenance in the target scenarios comprises various activities, we categorize the surveyed studies according to the taxonomy presented in Figure 8, which groups research efforts according to the following characteristics:

- **Target:** Components updated, repaired, or replaced during maintenance.
- **Strategy:** Maintenance strategies employed during the approached scenarios.
- **Technique:** Algorithms and methods used to implement the maintenance strategies.
- **Metric:** Performance indicators used to drive allocation decisions during maintenance.
- **Validation:** Evaluation methodologies used to assess the proposed solutions.

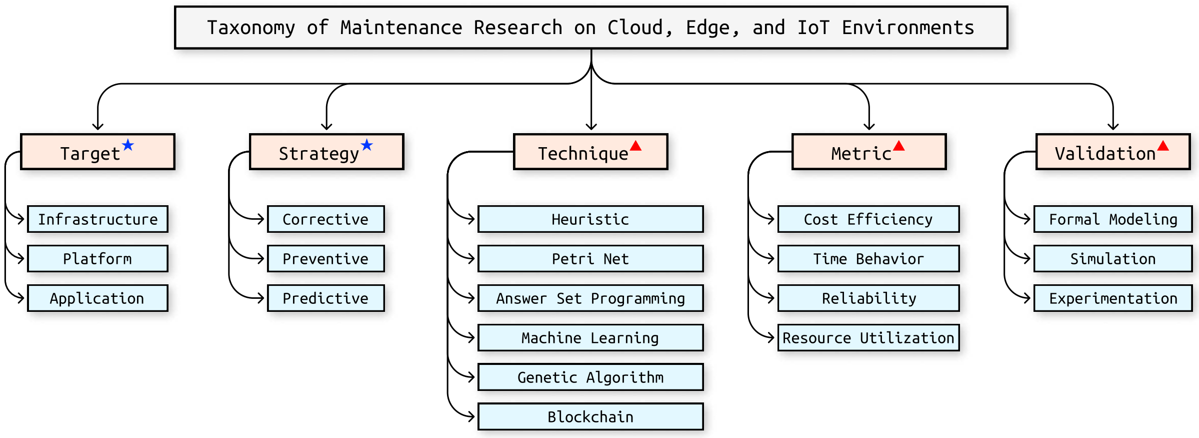

Fig. 8. Proposed taxonomy that organizes maintenance research aiming at cloud, edge, and IoT environments. A paper can fit in a single item of categories with the star mark (★) and in multiple items of categories with the triangle mark (▲).

The proposed taxonomy follows the feature-based branching model, as in Qu et al. [24] and Liu et al. [25], where a solution spans all the taxonomy categories. For example, a research paper can introduce a corrective maintenance approach (Strategy) based on Reinforcement Learning (Technique) capable of reducing migration time (Metric) during server updates (Target) in simulated (Validation) cloud data center scenarios. In this setting, each taxonomy branch discusses the selected studies from different perspectives, facilitating the categorization of solutions with common characteristics.

In the remaining of this section (§4.1–§4.5), we introduce existing maintenance strategies targeting cloud, edge, and IoT environments, categorizing them according to the branches of the proposed taxonomy.

4.1 Target

The "Target" category indicates the components manipulated in maintenance activities and the underlying concepts that comprise the theoretical foundations for proposed solutions. We divide maintenance targets into Infrastructure, Platform, and Application, as shown in Figure 9. The Infrastructure layer comprises low-level components, such as servers and networking components, which form the underlying structure for the computational environment. The Platform layer encompasses system-level software components, such as hypervisors and operating systems, which perform tasks such as management of computational resources and integration of hardware and applications. Finally, the Application layer groups end-user applications, which can be hosted in virtualized instances (e.g., VMs and containers) or run directly on their host operating system. It is worth noting that Figure 9 does not present an exhaustive list of the components included in each category. Instead, it encompasses a few examples from each category to demonstrate the employed rationale.

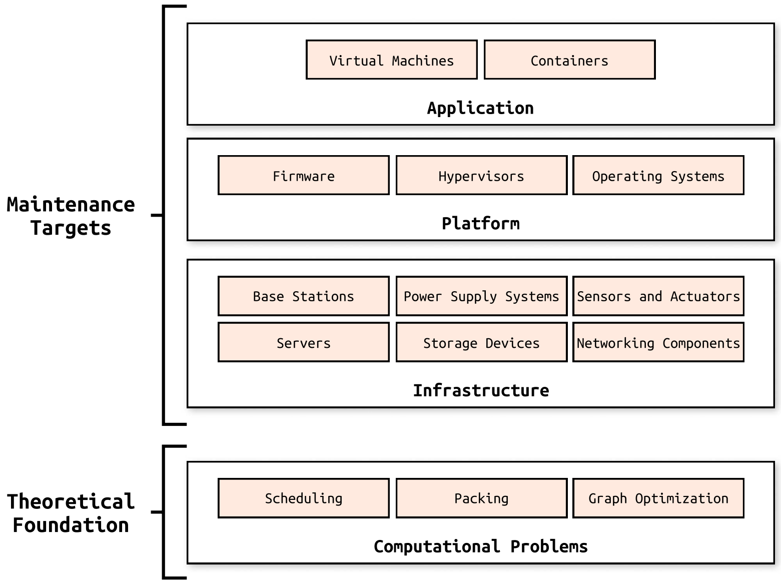

Fig. 9. Overview of components targeted during maintenance on cloud, edge, and IoT environments.

4.1.1 Theoretical Foundation. Considering the broad spectrum of maintenance scenarios in cloud, edge, and IoT infrastructures, existing research employs specific conceptual models to theoretically represent maintenance operations. Despite their diversity, most share two common characteristics. Firstly, it is assumed that computing infrastructures have limited resources, generally restricting the allocation decisions that can be made. Secondly, maintenance work is presumed to require the temporary suspension of target components and adjacent resources. Based on these premises, maintenance policies usually update or replace target components gradually to avoid application downtime. Such a design decision underscores the criticality of maintenance progression decisions and creates a natural link between maintenance modeling and scheduling problems, which are focused on the optimal allocation of resources to tasks over time while adhering to a set of predefined constraints [26].

Maintenance modeling broadly encompasses two main scheduling decisions: (i) determination of the order of application relocations (typically to evacuate resources needing maintenance) and (ii) decision about the update sequence for previously evacuated components. Decisions on application relocation scheduling often consider the workload intensity to avoid reallocations during periods of heavy load [27] and the application requirements to ensure an optimized packing of applications on available servers [28]. Complementarily, decisions on the component update order consider the available capacity [29], the energy consumption [30], and how strategically target components are located within the infrastructure [5] [8].

In addition to scheduling decisions, maintenance policies often must decide how applications are mapped across the infrastructure. At this stage, packing problems come into play. Packing problems relate to the challenge of packing objects into bins aiming, e.g., to use as few bins as possible [31]. In the context of maintenance modeling, packing problems are employed to model the mapping of applications across computing infrastructures undergoing maintenance, with objects representing applications and bins representing computing nodes.

Existing literature generalizes packing problems to provide static and dynamic representations of application mapping during maintenance operations. Static representations, also called application placement, define where applications will be provisioned (i.e., which computing nodes will accommodate them), not considering eventual changes in the infrastructure that may demand changes in application positioning. Conversely, dynamic representations, also called application relocation or migration, extend placement modeling to consider changes in where applications are hosted over time.

We observe that application migration is the predominant packing modeling approach in maintenance scenarios, as migration-based solutions allow for gradual updates to take place without sacrificing application availability. Nonetheless, some researchers also resort to placement modeling approaches within their maintenance policies. For instance, Wang et al. [32] implemented an application placement strategy to meet the needs of maintenance scenarios where applications cannot be relocated over time. To cope with such constraints, they assumed the possibility of provisioning multiple replicas for each application, thus ensuring that temporarily suspending some replicas to advance maintenance does not impact the overall application availability.

Despite the unique characteristics of cloud, edge, and IoT infrastructures, they share the need to maintain connectivity among infrastructure components. In this context, some researchers have applied graph theory concepts to their maintenance conceptual models, where the infrastructure is represented as a graph with nodes depicting computing and networking components and edges depicting the connectivity between them (either through physical links or wireless technology). In practice, the network resources (such as links, switches, and base stations) play a crucial role in operationalizing several resource allocation decisions made during maintenance operations, including the transfer of patches to infrastructure components and the relocation of applications to ensure service continuity.

The application of graph theory in maintenance modeling predominantly leads to the exploration of two types of graph problems: shortest path and maximum flow. Shortest path problems focus on identifying the shortest route between nodes in the graph [33] and are applied to represent and optimize the application latency in response to migration decisions made to progress maintenance work. Maximum flow problems address the challenge of maximizing the data flow that can be transported from a source to a destination node within the graph [34], thereby guiding allocation decisions that require transferring substantial amounts of data across the network. We observe that most solutions employing underlying concepts of maximum flow problems do so when dealing with application migrations [27] [29] [35].

4.1.2 Infrastructure. Wang et al. [36] focused on an Internet of Underwater Things (IoUT) scenario, where interconnected underwater sensors and actuators enable monitoring of hard-to-reach water areas. As IoUT infrastructures suffer from harsh communication between network nodes and various external factors such as water interference, conducting maintenance operations of IoUT nodes is very challenging. In such a scenario, the authors present an approach that employs Autonomous Underwater Vehicles (AUVs) to search for and repair faulty IoUT nodes.

Several research works have tackled server maintenance challenges in cloud and edge infrastructures. Nevertheless, most of these studies have focused on two specific challenges: (i) defining when servers undergo maintenance work and (ii) preserving application continuity during server maintenance. Whereas the former line of investigation directly addresses the challenge of performing server maintenance (e.g., in Okuno et al. [27]), the latter focuses on supporting server maintenance, ensuring it affects end-users as little as possible, typically by managing how applications are provisioned within the infrastructure before servers undergo maintenance (e.g., in Hou et al. [28] and Wang et al. [37]).

Some of the server maintenance studies argue that servers often must be rebooted for updates to take effect [28] [27] [32]. Therefore, proposed solutions employ rolling upgrade policies, in which servers are grouped into batches based on various criteria, and each batch is updated one at a time. Before being updated, servers undergo a draining process, where their applications are relocated to alternative hosts, avoiding application downtime due to server reboots. In this context, some efforts have proposed algorithms that orchestrate migrations during maintenance for various purposes, such as preventing network saturation [28].

Apart from migration planning to support server maintenance operations, some researchers, such as Okuno et al. [27], focused on the fact that routine service maintenance operations typically do not have strict deadlines. As such, these authors presented server maintenance scheduling policies that perform server maintenance during idle periods to reduce the impact of maintenance on applications. Other specific contributions include defining the application replica placement for scenarios where infrastructure operators cannot migrate applications during server updates due to regulatory constraints [32], allowing servers to undergo maintenance gradually whenever needed without necessarily affecting application continuity.

While routine maintenance commonly sets no strict completion deadlines, other events may require fast responses to minimize damage. Wu et al. [5] discussed the impact of power outages on the performance of applications in cloud data centers. In such a scenario, on the one hand, infrastructure operators must relocate applications hosted by the affected components as early as possible to avoid downtime. On the other hand, poorly planned migrations can prematurely drain the emergency power supply, leading to a complete data center blackout. In response to such a challenge, the authors presented two scheduling algorithms that move applications out from affected servers without unnecessarily increasing the infrastructure's power consumption.

The lack of compatibility among network appliances from different vendors and other manageability issues have paved the way for Software-Defined Networking (SDN) [38], which decouples control and data planes and gives more freedom to network operators. In a typical SDN infrastructure, controller nodes manage the entire network, defining, for example, the routing rules that guide packet-forwarding devices. Whenever routing rules change, controllers must broadcast them to all forwarding devices to ensure the network is up-to-date. Whereas such a centralized model grants fine-grained control, frequent rule changes can lead to significant network communication overhead. In response, some research efforts presented algorithms that schedule route update flows to avoid network saturation [39] [40]. Other specific contributions related to the maintenance of network services include repairing deprecated routing rules in scenarios with link failures [41] [42].

In addition to enabling improved manageability of routing rules, SDN provides the underlying features for Network Function Virtualization (NFV) [43], which replaces traditional hardware appliances by hosting Virtual Network Functions (VNFs) such as firewalls and intrusion detection systems on general-purpose servers. Maintenance research efforts targeting VNFs focused primarily on improving their fault tolerance. Raza et al. [44] presented a checkpoint and rollback VNF failure recovery strategy that regularly takes checkpoints and timely rolls back VNFs as failures are detected. Other studies also predict host failures and proactively provision VNFs on alternative servers to avoid downtime [45] [46].

While much of the research in infrastructure maintenance has focused on computing and networking devices, a few studies have also tackled storage maintenance challenges, explicitly performing storage data repair using erasure codes [47] [48]. For instance, Wu et al. [47] considered IoT scenarios composed of mobile nodes cooperating to process and store data. While node mobility enhances the network's flexibility, it raises concerns about fault tolerance, as nodes can move out of the cluster's coverage area, leading to the loss of data they hold. The authors tackled this challenge with a repair strategy based on erasure codes. Once they identify that a node is about to leave the cluster's coverage, they migrate and reconstruct that node's data on the other nodes in the cluster.

4.1.3 Platform. Most research in platform-level maintenance has concentrated on coordinating firmware patch distribution to IoT devices. As downloading the firmware patches is a prerequisite for starting the maintenance, some studies have presented routing strategies to avoid network bottlenecks and allow devices to download patches as soon as possible [49]. In contrast to the traditional update distribution model, where devices download firmware updates from predetermined nodes managed by vendors, several studies employed peer-to-peer architectures [50] [51] [52], where devices can download patches from neighboring nodes, avoiding saturation at specific points in the network and improving fault tolerance in case some of the nodes serving the updates fail. As such a collaborative ecosystem raises security concerns, some studies also focused on ensuring the integrity of downloaded patches, especially when they are fetched from neighboring nodes [53] [54] [55].

In addition to firmware update coordination, some studies in platform-level maintenance also focused on optimizing hypervisor updates. In this context, on the one hand, virtualization grants improved flexibility in managing computing resources; on the other hand, infrastructure operators must keep hypervisors up-to-date to preserve the environment's integrity and performance. One of the primary motivations for updating hypervisors is to avoid software aging, which occurs when hypervisors running for long periods start presenting bugs that can affect applications. Most software rejuvenation approaches resorted to migrating the running applications from aged hypervisors to healthy ones, which can be either co-located [56] or within different hosts within the infrastructure [57] [58].

While migration techniques allow infrastructure operators to rearrange applications within the infrastructure during maintenance, the number of simultaneous migrations is limited by the resulting network communication overhead. In addition, seamless migration often requires compatibility between hypervisors, narrowing the migration options on heterogeneous infrastructures. Alternatively, some studies focused on in-place hypervisor upgrade strategies, where patches are applied without evacuating servers. Segalini et al. [59] presented Hy-FiX, an in-place upgrade solution for Kernel-based Virtual Machine (KVM)[4] hypervisors. Hy-FiX combines two checkpoint techniques (*suspend-to-disk* and *suspend-to-RAM*), which dump VM data to disk but keep its state in memory for faster recovery. In addition, Hy-FiX employs a lazy initialization technique that reduces server reboot time, making in-place upgrades less intrusive for applications. Ngoc et al. [7] introduced a solution called HyperTP, which supports Xen[5] and KVM hypervisors and allows infrastructure operators to choose whether to migrate applications before updating hypervisors (out-of-place upgrade) or temporarily pause applications during the patching (in-place upgrade).

Some studies have also addressed the maintenance of platform-level components from a generic perspective [60] [29] [8] [30]. For instance, Souza et al. [8] argued that occasionally, edge servers must be rebooted for patches to take effect. Therefore, infrastructure operators need to define how to apply patches to the target servers and devise support plans for maintenance to ensure application continuity during the updates. The authors focused on rolling upgrade plans where hosts gradually undergo maintenance, enabling applications to be relocated to alternative hosts before their original hosts are updated and rebooted. Considering such a scenario, the authors presented two maintenance strategies that perform location-aware migration decisions, keeping applications near their users to avoid latency increases during server updates while not excessively increasing maintenance time.

A similar scenario was considered by Rubin et al. [30], which also concentrated on defining the application migration plan necessary to avoid service downtime during platform-level edge server updates. In addition to considering the application latency requirements when choosing alternative hosts for them, the authors argued the importance of incorporating power consumption awareness of edge servers during migration decisions to keep the edge infrastructure as efficient as possible. To address this scenario, the authors proposed a maintenance

[4]https://www.linux-kvm.org/page/Main_Page
[5]https://xenproject.org/

algorithm named Emma, which not only schedules the update of low-level software components of edge servers but also reallocates applications in a power-efficient manner, prioritizing alternative hosts with lower power consumption profiles. While virtualization techniques allow relocating applications with reduced downtime, Russinovich et al. [60] highlighted that these techniques yield a significant network communication overhead. Accordingly, they employed an approach that replaces relocation techniques by allowing servers to be rebooted faster and without discarding the state of running applications.

4.1.4 Application.

Saxena et al. [35] was the only research work to focus on the maintenance challenges of VM-based application deployments. The authors presented a maintenance algorithm that proactively identifies and mitigates VM resource starvation and degradation problems. In the addressed scenario, VM-based applications have two distinct properties that can make them susceptible to failure over time. First, they possess time-varying workloads, meaning they may become overloaded if their hosts do not have sufficient resources to cope with increasing demands. Second, they start to show degradation effects after being active for an extended period, which eventually causes them to fail. The proposed algorithm deals with such a scenario by proactively identifying failure-prone VMs (both due to resource starvation and degradation) and triggering migration and replication routines that provision vulnerable VMs to better-suited hosts before the failures occur.

Olorunnife et al. [61] highlighted that containerization's lightweight capabilities facilitate software maintenance by allowing quick application reconfiguration. Nevertheless, the authors argue that reconfiguration becomes ineffective if local container image metadata is corrupted — in such cases, rebuilt containers will also present undesired behaviors. Therefore, the authors presented a framework that tackles software failures caused by runtime changes in containerized applications. Whenever the proposed framework identifies a failing application, it rebuilds the affected container based on the metadata of healthy nodes, discarding any settings that might have caused the failure.

Despite a few maintenance initiatives focused on addressing maintenance challenges specific to VM and container deployments, most existing literature handles software maintenance from a general perspective without specifying the application deployment specifications. Among the research works focused on maintenance challenges in generic application deployment scenarios, Weißbach et al. [62] and Bui et al. [63] proposed patch distribution scheduling strategies that consider software dependencies that add constraints regarding the update order. Additionally, some studies have also considered the security implications of software patch distribution in peer-to-peer networks [64] [65] [66], where choosing isolated nodes to distribute patches can delay maintenance, and compromised nodes can propagate malicious software. Other investigations have also concentrated on repairing hang bugs [67] and performing maintenance operations to avoid aging-related software issues [68].

4.1.5 Summary.

Existing research works covers repairing, upgrading, and replacing various physical and logical components in cloud, edge, and IoT environments. Most research efforts targeting the maintenance of physical components focused on cloud data centers considering hyper-converged infrastructures [69] [70], which replace storage appliances with servers that bundle both compute and storage capabilities. Consequently, most strategies were designed for conducting server maintenance. As for the research on the maintenance of logical components, we can observe specific directions followed by cloud and IoT papers. While cloud research focused on hypervisor maintenance, most IoT research optimized resource usage, cost, and security during patch distribution. The few papers targeting logical components in edge sites presented maintenance strategies for VNFs and general-purpose applications. Table 3 summarizes the maintenance targets covered in the selected papers.

4.2 Strategy

The "Strategy" category discusses the role of the different maintenance approaches presented in the taxonomy of Figure 2 (i.e., corrective maintenance, preventive maintenance, and predictive maintenance) in research efforts

Table 3. Summary of maintenance targets covered in the selected papers.

Target		References
Application	VM Deployments	[35]
	Container Deployments	[61]
	Generic Deployments	[62] [68] [63] [64] [65] [67] [66]
Platform	Firmware	[71] [54] [72] [49] [55] [53] [51] [73] [50] [6] [52] [74] [75] [76] [77] [60] [29] [8] [30]
	Hypervisors	[57] [58] [59] [56] [7] [60] [29] [8] [30]
Infrastructure	Base Stations	[78]
	IoT Devices	[36]
	Power Supplies	[5]
	Servers	[27] [32] [28] [37]
	Storage Devices	[48] [47]
	Virtual Network Functions	[44] [45] [46]
	Routing Rules	[41] [39] [42] [40]

for cloud, edge, and IoT environments. In addition to summarizing the commonalities between solutions under the same maintenance strategy, we present a holistic view that examines the pros and cons of each approach regarding the various maintenance demands in the addressed paradigms.

4.2.1 Corrective Maintenance. Most existing maintenance research efforts concentrated on corrective strategies, either acting upon failure or delegating the decision of "when" to initiate maintenance to third-party entities or events (e.g., the release of security patches or the arrival of new components to replace existing ones). Consequently, proposed solutions have mainly focused on optimizing "how" maintenance takes place, which comprises several decisions, such as defining strategies to reduce maintenance's impact on applications [59] [7] and the component update order when they cannot be updated at once (typically due to QoS constraints) [8] [29].

While some corrective maintenance strategies can advance or postpone maintenance work on convenience in scenarios without strict deadlines, they act on the premise that the decision to carry out maintenance has already been taken, solely deciding the best schedule for it [27] [29]. In such cases, the decision to perform maintenance often comes from external sources such as anomaly detection algorithms, which analyze logs and performance metrics to identify problematic events [79]. In computing infrastructures, anomaly detection algorithms rely on monitoring systems (e.g., Zabbix[6], Nagios[7], and Instana[8]) that collect data through monitoring agents installed in the hardware and various software layers (hypervisor, operating system, and applications).

Despite growing evidence about the potential of proactive strategies [23], there is a concern regarding prediction errors, as maintenance work performed unnecessarily can quickly saturate the infrastructure and degrade application performance. Consequently, corrective maintenance has remained in sight of the research community, especially due to the potential risks incurred by some events that are hard to predict (e.g., power outages [5] and certain cyberattacks [58]). In such cases, maintenance strategies have mainly focused on mitigating the problem that triggered the maintenance as efficiently as possible rather than taking the risk of making wrong maintenance decisions based on inaccurate feedback from proactive approaches.

4.2.2 Preventive Maintenance. While corrective maintenance is effective in some scenarios, preventive approaches can reduce maintenance costs when handling issues with deterministic behavior, such as software aging. Preventive maintenance strategies mitigate software aging through software rejuvenation techniques, which typically

[6]https://www.zabbix.com/
[7]https://www.nagios.org/
[8]https://www.instana.com/

involve gracefully restarting software components to clean up their internal state [80]. As discussed in Section 2.4, preventive maintenance comprises schedule-based and condition-based approaches. While schedule-based maintenance strategies do not require real-time monitoring mechanisms, finding appropriate maintenance intervals is challenging in some situations. On the other hand, condition-based maintenance strategies grant enhanced flexibility but assume that degradation advances slowly enough to enable on-point correction. Accordingly, existing preventive maintenance strategies employ schedule-based and condition-based policies on convenience, often combining them when needed.

Fakhrolmobasheri et al. [57] discussed the criticality of hypervisor rejuvenation in virtualized infrastructures, as hypervisor failures intrinsically affect all hosted VMs. In such a scenario, aging occurs due to several factors, such as the lifetime and workload pattern of hosted VMs. The authors employed a condition-based maintenance approach that avoids aging-related failures in hypervisors through a two-threshold policy. Whenever a hypervisor indicates slight aging, the first threshold is triggered and hosted VMs are migrated if the hosts of other hypervisors have enough available resources. However, as migrations are not mandatory, a hypervisor can reach an advanced aging level where failures are more likely to happen before its VMs are relocated. In that case, the second threshold is triggered, and hosted VMs enter the queue for immediate migration regardless of the occupation of alternative hosts, as their hypervisor must be evacuated for rejuvenation as early as possible. The proposed solution also performs migrations to consolidate VMs on the most occupied hosts, switching off idle servers to reduce power consumption and reverse any aging effects of hosted hypervisors, which makes hypervisors ready for later usage if their servers are restarted.

Meng et al. [68] tackled aging-related issues more broadly, discussing the role of software rejuvenation in preserving the health of general-purpose applications. The authors described that aging progression damages components until it leads to functional failure. In such a scenario, if threshold-based maintenance policies manage to detect aging properly, rejuvenation is eventually triggered, resetting the accumulated damage. However, if the aging progression pattern is unknown to the thresholds, maintenance only takes place in a corrective fashion after the software failure. After discussing the risks of relying solely on threshold-based maintenance policies, the authors implement a hybrid strategy that triggers software rejuvenation at certain damage thresholds or predefined intervals, whichever occurs first. Accordingly, even failures displaying unknown progression that could bypass the thresholds are mitigated by maintenance work performed at fixed intervals.

4.2.3 Predictive Maintenance. Preventive maintenance strategies rely on maintenance intervals (schedule-based maintenance) and damage thresholds (condition-based maintenance), which might be hard to define in some scenarios. In addition, preventive maintenance requires certain assumptions to be true to function correctly. While schedule-based maintenance assumes that damage progression is uniform so that failures do not occur between maintenance runs, condition-based maintenance assumes that damage progresses slowly enough so that the intervals in which thresholds are triggered and functional failures are sufficiently long for maintenance personnel to act. As such assumptions may not be satisfied in certain scenarios, some research works leverage predictive maintenance approaches to forecast the upcoming system state and take more effective measures.

Liu et al. [47] focused on increasing disk fault tolerance in clusters of mobile IoT nodes (e.g., smart vehicles and drones). In such a scenario, mobile nodes can quickly leave the cluster's coverage area, causing loss of the data they store. The authors presented two proactive data repair techniques based on predictions that indicate nodes about to leave the cluster's area (so-called soon-to-fail nodes). Specifically, the first technique replicates data from soon-to-fail nodes into healthy nodes through migration over the network, and the second technique proactively reconstructs data chunks from several healthy nodes through erasure codes. While replication and erasure codes optimize various aspects within the system (i.e., avoiding single points of failure and reducing disk usage), the authors do not detail how node mobility prediction happens, delegating this feature to third-party entities.

Huang et al. [45] discussed the challenges of providing fault-tolerant VNFs in edge infrastructures. In such a scenario, the authors argued that the potential failure surface at the edge spans physical and logical layers in the edge servers and network components such as switches and links. In addition, acting reactively upon failure incurs a high operational cost as VNFs provide core underlying features for end-user applications (e.g., firewall, load balancing), and the lack of these features even for a short period can lead to significant security and performance issues. To address these issues, the authors presented a proactive VNF failover architecture that details the requirements and core components to provide resilient VNF services at the edge. In addition, Huang et al. [46] presented a novel algorithm that predicts network failures and proactively reprovisions VNF instances on unaffected components to avoid service outages.

Saxena et al. [35] focused on ensuring high availability and fault tolerance for general-purpose applications in cloud data centers. The authors argued that resource saturation represents a significant portion of service outages in cloud environments, often occurring due to unexpected workload variations and over-extended time needed for reprovisioning applications. Therefore, they introduced a resource utilization forecast mechanism that anticipates server overutilization. The proposed solution categorizes applications into two groups, normal and failure-prone, depending on whether their servers have sufficient resources for the near future or they are about to be overloaded, respectively. Then, the resource usage predictions feed migration and replica placement policies that provision failure-prone applications on alternative servers to avoid service outages and improve the data center's resource efficiency.

4.2.4 Summary. Existing maintenance research efforts employed different approaches depending on the addressed scenarios. Corrective solutions typically offload the decision of when to start maintenance to third-party entities (e.g., external analytics tools), focusing on components with low repair costs without strict maintenance completion deadlines or when maintenance triggers are caused by hard-to-predict events (e.g., security patch releases). We can also observe that preventive maintenance is favored when maintenance triggers display well-defined patterns (e.g., software aging). Finally, predictive maintenance is often employed when proactiveness is a requirement, and monitoring indicators move too fast for preventive approaches to be used. In such cases, predictive methods can forecast the system state to support infrastructure operators with actionable insights and sufficient time to work. Table 4 summarizes the maintenance strategies employed by the selected papers.

Table 4. Summary of maintenance strategies covered in the selected papers.

Strategies		**References**
Corrective		[67] [61] [62] [63] [64] [65] [66] [78] [36] [5] [27] [32] [28] [44] [71] [54] [72] [49] [55] [53] [51] [73] [50] [6] [52] [74] [75] [76] [77] [58] [59] [56] [7] [60] [29] [8] [30] [41] [39] [42] [40]
Preventive	Schedule-Based Maintenance	[68]
	Condition-Based Maintenance	[68] [57] [37]
Predictive	Data-Based Maintenance	[35] [48] [47] [45] [46]

4.3 Technique

The "Technique" branch classifies the methods and algorithms employed within surveyed maintenance strategies. As some studies propose conceptual maintenance architectures instead of algorithms for performing maintenance, we put them into the "Heuristic" category. To this end, we consider heuristic any procedure that adopts rules of thumb to solve a particular problem and, within the scope of this review, that also does not fit into the other discussed methods and techniques.

4.3.1 Heuristic. As maintenance encompasses the coordination of several complex decision-making processes, most of the strategies in the literature employ heuristic procedures to obtain sufficiently good solutions in a reasonable time. Maintenance heuristics generally incorporate groups of predefined rules that trigger various actions during maintenance. Despite the significant differences among solutions due to the varying requirements of addressed scenarios, maintenance heuristics often have similarities, such as the adoption of sorting policies based on custom cost and score functions to make decisions such as scheduling application migrations and defining the components update order [28] [71]. In addition, some heuristic solutions also make more specific decisions, such as identifying maintenance trigger events such as software bugs [67].

While most maintenance heuristics implemented problem-specific procedures to make accurate decisions, some proposals employed ensemble approaches, which combine multiple techniques to cope with broader scenarios or to increase the solution's robustness (e.g., targeting global optima rather than local optima). Saxena et al. [35] proposed an ensemble maintenance heuristic called OFP-TM. While OFP-TM follows a set of predefined rules during allocation decisions to improve the fault tolerance of cloud applications, it employs Artificial Intelligence models to proactively identify failure-prone applications.

4.3.2 Petri Net. The need to represent the variety of simultaneous processes and events during maintenance has drawn the community's attention to modeling techniques such as Petri Networks (Petri Nets) [81], which provide means for graphically modeling complex processes. Petri Nets comprise groups of places, transitions, and arcs. While places and transitions denote system states and events that move the system from one place to another, respectively, arcs represent relationships between places and transitions. In this setting, tokens denote the Petri Net workflow as its transitions are triggered. Some studies leveraged Petri Nets to model dynamic maintenance systems, where multiple events can occur simultaneously, subject to precedence and frequency constraints. Specifically, we observe an interest in Petri Nets to model maintenance scheduling algorithms in software rejuvenation scenarios, where concurrent events such as user requests directly affect how fast component aging progresses.

Fakhrolmobasheri et al. [57] proposed a hypervisor rejuvenation system based on Stochastic Activity Networks (SANs) [82], an extension of Petri Nets that facilitates the modeling of activities such as user requests whose duration impacts system performance. The proposed SAN model comprises five sub-models representing the arrival of concurrent user requests, application migrations between hypervisors, a shutdown mechanism for idle servers, hypervisor rejuvenation, and aging-related failures. In this setting, several functions determine appropriate times to rejuvenate the hypervisors to mitigate software failures while reducing the infrastructure's power consumption. Torquato et al. [58] followed a similar line of reasoning, employing Stochastic Rewards Networks (SRNs) [83], which extend SANs with reward-based functions for measuring the reliability of complex systems. The authors presented a maintenance system that schedules application migrations to rejuvenate hypervisors while reducing the vulnerability surface of applications against network attacks.

4.3.3 Answer Set Programming. Maintenance modeling typically encompasses several problems with high computational complexity, such as application migration and process scheduling. As a result, finding optimal solutions to large-scale maintenance problems within a reasonable time is often unfeasible. Although heuristics are typically used as an alternative approach, defining appropriate rules of thumb for such strategies might be challenging in some maintenance scenarios with several components and dynamic behavior. In this context, Answer Set Programming (ASP) [84] comes into the spotlight as an efficient alternative that, unlike traditional programming paradigms, which require explicit search instructions, employs Answer Set Solvers to efficiently find solutions through the definition of an objective function and constraints that must be satisfied for a solution to be considered valid. In this way, ASP avoids suboptimal solutions resulting from inaccurate algorithmic instructions.

Okuno et al. [27] employed ASP to formulate a cloud server maintenance problem focused on optimizing two conflicting goals: preserving application availability and reducing maintenance time. To this end, the proposed model tries to minimize maintenance time while a set of constraints restricts the migration scheduling during periods where applications are under heavy load. Given the high complexity of defining the optimal maintenance scheduling in such a scenario, the proposed solution employs a Divide-and-Conquer approach that partitions the maintenance scheduling into subproblems composed of groups of servers and time slots. After solving each subproblem individually, the proposed solution combines the partial solutions, obtaining the optimal maintenance schedule for the entire data center. The authors demonstrated that the proposed solution finds optimal maintenance schedules while displaying reduced time and space complexity compared to the baseline approach that employs ASP without the proposed divide-and-conquer strategy.

4.3.4 Machine Learning. Efficient maintenance planning requires extensive effort, even for domain experts, as resource allocation decisions during maintenance (e.g., component update order definition or application migration scheduling) can lead to undesirable cascading events that are difficult to track. This problem becomes even more challenging for the maintenance of critical assets, where proactive decisions are required to reduce repair costs and avoid service disruption. In this context, some research efforts advocated the usage of Machine Learning (ML) [85] models to obtain accurate maintenance decisions. Unlike traditional algorithms, ML models can discover hidden patterns and correlations in input data to identify ongoing events or predict upcoming environment states.

Several ML algorithms learn how to approach target problems through a training phase, building their internal rules through identified patterns in historical data. In this context, ML learning algorithms can define biased rules if their starting search point in the training dataset leads to local optima. To overcome such challenges, a branch of ML called Ensemble Learning [86] combines the output of multiple ML algorithms. The diversity of Ensemble methods often grants them higher accuracy than individual models, as they are not restricted to the decisions of a single learning algorithm. Saxena et al. [35] presented an Ensemble algorithm that combines three ML models (Feed-Forward Neural Network, Support Vector Machine, and Linear Regression) to predict failures in cloud applications. Similarly, Huang et al. [45] employed a tree-based ensemble algorithm called Random Forest to predict failures that could affect VNFs in edge infrastructures.

Although ML models display great potential for handling non-trivial optimization problems, they typically require significant training data. Consequently, ML models show limitations when input data frequently changes or when there is no historical data for training. In response, Reinforcement Learning (RL) [87] emerged as a branch of ML that employs intelligent agents capable of adjusting their internal rules on-the-fly based on reward and penalty systems.

Motivated by the benefits of RL algorithms, Nain et al. [40] employed this type of model to tackle the challenge of coordinating the distribution of routing rules in SDN-supported IoT environments. In such a scenario, while indiscriminately broadcasting updated routing rules to nodes incurs network saturation, excessively delaying the distribution of new rules may expose the network to transmission failures and security vulnerabilities. Accordingly, the authors employed an RL algorithm called Q-Learning, which dynamically coordinates the distribution of updated routing rules to ensure that nodes are up-to-date as early as possible without causing network saturation. Targeting a different scenario, Ying et al. [29] used RL to schedule application migrations during maintenance in cloud data centers where there is no previous knowledge about the network state (e.g., link delays, bandwidth usage, etc.).

RL stands out against traditional ML by implementing agents that can adapt and learn from their own experiences. Despite their benefits, typical RL models follow a single-agent structure, so they are not optimized for problems requiring social abilities such as cooperation and reciprocity. Based on this rationale, Wang et al. [36] explored the use of a subfield of RL known as Multi-Agent Reinforcement Learning (MARL), where multiple agents learn simultaneously within a shared environment. As MARL models implement an ecosystem where the actions of one agent can affect the outcomes and strategies of others, their agents can learn to interact and coexist with their peers despite having their individual objectives. Bringing the benefits of MARL to the maintenance landscape, Wang et al. [36] designed a MARL model where each agent represents an underwater vehicle that must search and repair failing nodes in Internet of Underwater Things (IoUT) scenarios. The underwater vehicles were designed to cooperate, sharing known map information to optimize the search and start the maintenance of failing IoUT nodes as soon as possible.

4.3.5 Genetic Algorithm. Like declarative paradigms and ML models, metaheuristics are heuristic replacements that define high-level and problem-independent procedures for tackling complex optimization problems. One of the most known metaheuristic methods is called Genetic Algorithm (GA) [88], which mimics the biological process described by Darwin's theory of evolution by natural selection. Rather than evolving a single solution at a time, GAs employ a population-based search strategy that evolves multiple solutions simultaneously according to predefined objectives. One of the main features of GAs is the use of mutation and crossover mechanisms to balance exploration and exploitation when looking for efficient solutions. While exploration suggests visiting new regions in the search space (often far from explored points), exploitation suggests visiting the surroundings of already explored regions.

Souza et al. [8] employed a multiobjective GA called Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [89] to conduct server maintenance in edge infrastructures while considering application latency requirements. Unlike traditional GAs, NSGA-II is designed to find Pareto-optimal solutions for multiobjective optimization problems through an elitist approach that favors non-dominated solutions. As the authors adopted a batch-based maintenance model where the servers drained in a given batch are updated in the next, the application migration schedule affects the server update order. Consequently, the proposed solution finds the best application migration scheduling to reduce maintenance time without neglecting end-user latency requirements.

4.3.6 Blockchain. Patch distribution is a vital maintenance process that involves transferring updated binaries over the network from vendors to outdated nodes. One of the main challenges regarding patch distribution is ensuring the integrity of patches against potential malware injections in the network. Patch distribution security is even more concerning in peer-to-peer infrastructures, where the attack surface goes beyond the network, as compromised nodes may distribute malware to neighboring nodes. In this context, distributed ledger technology such as Blockchain [90] comes into the spotlight as distributed databases that store transaction records in multiple nodes simultaneously, enabling the implementation of consensus mechanisms to prevent malicious changes during patch distribution.

Several research works have proposed solutions using Blockchain technologies such as Bitcoin[9], Hyperledger Fabric[10], and Ethereum[11] for secure patch distribution. Most proposals presented frameworks that leverage the Blockchain's block validation process, where individual transactions (e.g., patch submissions) are verified before being added to the permanent ledger of the Blockchain, which reduces the chances of malware distribution on the network [55] [73] [75] [74] [72] [51] [65]. As Blockchain security increases as more nodes join the network, some research efforts also presented incentive strategies that compensate nodes for participating in the Blockchain network as patch distributors [49] [50] [52].

4.3.7 Summary. Existing maintenance solutions use multiple technologies depending on the addressed scenarios. Most proposed algorithms employed heuristic procedures to increase flexibility and reduce prototyping speed. We observe a clear preference for ML to develop predictive approaches. In addition, Petri Nets serve as good alternatives for graphically modeling complex maintenance systems, while GAs and ASP facilitate the development of maintenance strategies for scenarios where it is challenging to define algorithmic procedures to find solutions.

[9]https://bitcoin.org/
[10]https://www.hyperledger.org/
[11]https://ethereum.org/

Finally, most approaches focused on distributing firmware to nodes in IoT environments used Blockchain technology for enhanced security, which is a vital objective as IoT devices are typically installed outdoors, which increases the risk of physical tampering, and connectivity is often provided by public wireless networks, which widens the attack surface. Table 5 summarizes the techniques employed by the studied maintenance strategies.

Table 5. Summary of techniques used by the strategies proposed in the selected papers.

Techniques		**References**
Heuristic		[59] [56] [60] [7] [68] [67] [42] [44] [32] [5] [35] [8] [63] [64] [77] [48] [62] [53] [71] [6] [41] [61] [78] [39] [28] [54] [46] [66] [47] [30]
Metaheuristic	Genetic Algorithm	[8]
	Sine Cosine Algorithm	[37]
Petri Net	Stochastic Reward Network	[58]
	Stochastic Activity Network	[57]
Answer Set Programming	ASP-Core-2	[27]
Machine Learning	Neural Network	[35]
	Support Vector Machine	[35] [45]
	Linear Regression	[35]
	Reinforcement Learning	[29] [40] [36]
	Random Forest	[45]
Blockchain	Bitcoin	[50]
	Hyperledger Fabric	[55]
	Ethereum	[49] [51] [52] [65] [72] [75] [74] [76]
	Custom Blockchain	[73]

4.4 Metric

The "Metric" category organizes and discusses the metrics adopted as performance indicators in the evaluation of proposed maintenance strategies. To provide a logical organization of the analyzed metrics, we grouped them according to four expressed properties: Cost Efficiency, Reliability, Time Behavior, and Resource Utilization. While Cost Efficiency was included due to the significant number of papers that employed monetary and theoretical cost measures as the principal performance measure in their evaluations, Reliability, Time Behavior, and Resource Utilization are formal categories defined within the quality model of the ISO/IEC 25010 standard[12]. It is worth noting that a single metric (or variations of it) can be used to evaluate several performance facets, given the wide variety of maintenance scenarios addressed in the reviewed papers.

4.4.1 Cost Efficiency. Most maintenance research works that employed cost metrics are related to Blockchain's usage. In Blockchain networks, miners play a critical role in preserving network security by validating each transaction before adding it to the Blockchain ledger [90]. As an incentive, miners get rewarded for the computational effort incurred by transaction validation through network fees charged to Blockchain users [91]. In addition to incentivizing miners' work, fees help prevent spam attacks, which become costly to implement at scale. Although network fees are vital in the Blockchain's security ecosystem, some maintenance papers evaluated the cost efficiency of proposed Blockchain systems as indiscriminately inflated fees can hinder Blockchain's sustainability [49] [65] [73] [52] [75] [74].

[12]https://www.iso.org/standard/35733.html

In this line of reasoning, Tapas et al. [50] presented P^4UIoT, a firmware update distribution system that employs Bitcoin's Lightning Network[13] to increase the network throughput during patch distribution without excessively raising network fees. The Lightning Network creates payment channels between parties in the network, allowing cheaper and faster transactions to be exchanged outside the main Blockchain. Once the payment channel is closed, off-chain transactions are consolidated and broadcast to the main Blockchain. As just a single transaction resulting from the payment channel is added to the main Blockchain ledger, the Lightning Network enables reduced traffic on the Blockchain and lower network fees.

Huang et al. [46] deviated from the Blockchain's scope, employing cost as a performance indicator during the maintenance of failing VNF deployments comprised of a master and multiple backup instances. While failures in the master VNF instance require selecting a new master among the backup instances and updating the routing path accordingly, failures in the backup instances solely require deploying replacement backup instances. In this context, the authors presented a failure recovery cost function that considers the VNF recovery time during failures in the master and backup instances.

4.4.2 Reliability. Metrics within the Reliability property indicate the degree to which the components from the maintained environments can function under the expected conditions. Reliability metrics are valuable indicators for maintenance strategies as they measure both the effectiveness of maintenance work (e.g., in terms of the number of successfully recovered components) and its impact on the functioning of applications (e.g., in terms of availability and failure probability).

He et al. [67] concentrated on the repair of hang bugs, which are difficult to identify as they undermine the system's responsiveness without causing explicit failures. Considering this, the authors evaluated the effectiveness of repair strategies regarding the number of partially and completely fixed hang bugs. Similarly, Fakhrolmobasheri et al. [57] assessed the impact of hypervisor rejuvenation strategies in preventing software failures. During their evaluation, the authors employed several Reliability metrics, such as the number of hypervisor failures, the ratio of serving VMs to accepted requests, and the number of VM failures. Torquato et al. [58] evaluated the impact of migrations performed to mitigate hypervisor aging on the application's availability. Other research efforts discussed the potential downsides of application migration more generically, highlighting their implications on application availability [35] [28].

4.4.3 Time Behavior. Metrics within the Time Behavior property indicate the execution time of maintenance strategies and the impact of maintenance work on the response time, processing time, and throughput rates of applications. As such, they allow infrastructure operators to observe whether maintenance strategies can provide timely operational plans in compliance with user performance requirements.

Okuno et al. [27] evaluated the time needed to define the schedule of cloud server rolling updates. As looking for optimal solutions can become impractical due to the large search space, the proposed maintenance scheduler employs a divide-and-conquer approach that extends an ASP-based model to find quality solutions within a reasonable time. The authors evaluated their scheduler's execution time in different scenarios with a varying number of servers and applications, comparing the baseline ASP model to the proposed divide-and-conquer approach. During the evaluation, they also measured the time taken by the divide-and-conquer approach to solving the smallest and largest scheduling subproblems, identifying possible performance bottlenecks. In a similar line of reasoning, Bui et al. [63] employed planning runtime as a performance indicator during the maintenance of composite IoT applications. The authors compared the running time of a proposed heuristic against the CPLEX mathematical solver in five scenarios with different numbers of devices, demonstrating that the proposed heuristic could find near-optimal solutions orders of magnitude faster than the CPLEX solver.

[13]https://lightning.network/

Banikhalaf et al. [41] evaluated application delay during route repairs in vehicular networks, where wireless connectivity and vehicle movement amplify the potential impact of improper route changes. He et al. [55] approached a different scenario, evaluating the delay of transactions that check the integrity of firmware patches on Blockchain networks, in which increased response times can result from bottlenecks both on the network and on computing nodes. Other efforts evaluated the impact of network-hungry operations such as patch distribution [72] [51] and routing rules propagation [40] [42] on the application throughput.

In addition to analyzing the time required to obtain operational maintenance plans and the impact of maintenance work on application performance, some research works evaluated the quality of the proposed solutions regarding the time needed to complete the maintenance. Overall, maintenance time was assessed based on two approaches: (i) the number of batches and (ii) the aggregated duration of events that occur during maintenance. Maintenance modeling generally encompasses several events that can occur simultaneously and potentially affect each other. As representing such interactions incurs high computational effort at scale, batch-based maintenance modeling emerges as a lightweight alternative. Although the batch-based model allows measuring the duration of specific events during maintenance (see Souza et al. [8] for example), some studies employ simplified models where maintenance time is evaluated by the number of batches needed to update all target components. Following this reasoning, Hou et al. [28] used batch-based modeling to evaluate time efficiency during edge server updates. In such a scenario, each batch incorporates the selection of which servers will undergo maintenance and the migration of applications in those servers to alternative hosts before the update starts.

Although the number of batches provides a notion of time, most research efforts evaluated maintenance strategies by unraveling maintenance time according to the duration of various events, such as application migrations, recovery, patching, and other specific operations. There are different approaches to measuring update time depending on the scenario and employed techniques.

While some investigations discussed generic benefits of reduced update times, such as performance improvements [8], other works indicated specific gains, such as reduced attack surface in the case of security patches [7] [71], and shortened service disruption in the case of failure correction updates [61] [47] [35] [67] [60]. Specific update time measures are seen in some maintenance scenarios, such as in-place hypervisor updates, where the time for storing/restoring application states and restarting the servers is also considered [59] [7] [60]. Other operations comprised within the maintenance time include the computations performed to decode encrypted patches [54] [64] and the time required to release patches on Blockchain networks [65] [73].

4.4.4 Resource Utilization. Metrics under the Resource Utilization property denote the amounts and types of resources consumed by maintenance operations in the target environment. These are valuable indicators given that maintenance encompasses several activities that can saturate network and compute resources.

Network resource utilization was predominantly assessed during broken link recovery in SDN infrastructures, where control packets with updated routing rules must be distributed to forwarding devices to isolate failing resources and avoid packet losses. In such scenarios, decisions disregarding the dispatch order and the bandwidth reserved for propagating control packets can render damaging effects such as network starvation. Accordingly, resource efforts targeting this scenario evaluated the resource utilization of maintenance strategies regarding the number of links and bandwidth used during the control packets distribution to identify over-consuming approaches [39] [41]. As for compute node resource utilization, there is a preference for reducing memory consumption during the maintenance of low-level applications (i.e., firmware and hypervisors). While firmware update approaches aimed at minimizing memory usage in operations such as patch propagation, downloading, decompression, integrity checking, and update installation [53] [6] [75], hypervisor repair papers concentrated on minimizing the memory requirements of in-place upgrades and application migrations [59] [7].

Apart from raw compute and network usage, power consumption was also considered for evaluation of resource utilization in specific scenarios where energy-draining operations could impact service continuity and sustainability goals. Wu et al. [5] discussed the impact of migration decisions on energy efficiency during the evacuation of data centers under power outages. Although data centers typically have abundant power supplies, this situation is reversed during power outages, where infrastructure operators must define application migration plans to evacuate the affected data centers while spare power supply systems are still working. In such a scenario, inefficient migration decisions can drain the backup energy supply and cause various issues, such as data loss and hardware failures due to improper halting.

Fakhrolmobasheri et al. [57] discussed how large-scale cloud data centers represent a substantial portion of society's electricity usage and how this impacts greenhouse gas emissions. As such, the authors included power efficiency as a maintenance target for hypervisor rejuvenation work. Other research efforts focused on minimizing maintenance power consumption in IoT environments, where expensive operations such as Blockchain computations and network transfers can quickly drain the battery life of computing devices [75] [73] [42].

4.4.5 Summary. Maintenance typically aims to mitigate or prevent critical issues such as security vulnerabilities and infrastructure failures, where poor decisions can lead to severe consequences. As such, the quality of maintenance strategies is often measured through cost metrics that consider several factors, from monetary charges to failure costs. Several research efforts also evaluate maintenance strategies regarding time-related metrics such as patching time and the number of batches/rounds required to complete the maintenance.

While maintenance activities help preserve the performance and security of target environments, they encompass several resource-intensive activities that can cause harmful effects on the infrastructure if not accompanied by proper planning and execution. In this context, resource utilization and application performance metrics are also considered to assess the feasibility and efficiency of maintenance strategies. Whereas resource efficiency includes the demand of compute nodes (i.e., CPU, memory, and disk), the network occupation (ingress/egress traffic and the number of used links), and the infrastructure's power consumption, quality of service is typically represented through network-related metrics (e.g., delay and throughput) and service continuity (e.g., availability and the number of failures). Table 6 summarizes the properties and metrics used to evaluate maintenance strategies.

4.5 Validation

The "Validation" category indicates the validation approaches used to evaluate proposed maintenance solutions. We divide the validation approaches into three groups: (i) formal modeling, (ii) simulation, and (iii) empirical experimentation. While simulation is self-explanatory, formal modeling comprises the adoption of mathematical models and formal analyses, and empirical experimentation implies conducting experiments on practical testbeds.

4.5.1 Formal Modeling. Formal methods enable the analysis and evaluation of research prototypes without requiring practical implementations, which allows cost savings related to building and maintaining real testbeds or developing simulation tools. Hu et al. [72] employed a formal analysis to validate their approach for securely delivering firmware updates to IoT devices. The authors introduced several theorems demonstrating their system's effectiveness in preventing various cyberattacks, such as denial of service and the distribution of malware pretending to be regular firmware updates. In addition, they presented a computational complexity analysis to compare the proposed solution against an approach from the literature. Okuno et al. [27] also employed a formal method during their evaluation, representing the addressed maintenance scenario with *clingo* [92], an ASP system that enables the modeling of large-scale combinatorial problems as logic programs.

4.5.2 Simulation. While formal modeling methods are suitable for early-stage research projects, they typically require the adoption of many high-level abstractions to find reasonable solutions in a feasible time. In this context, simulation comes into the scene as an alternative that allows rapid prototyping with more realistic conceptual models and lower implementation costs. Most existing research efforts that present simulation-based evaluations employed custom simulators [68] [5] [71] [28] [39] [63] [65] [45] [32] [46] [35] [47] [29]. Although this facilitates

Table 6. Summary of properties and metrics considered in the evaluation of proposed maintenance strategies.

Properties	**Metrics**	**References**
Cost Efficiency	Monetary Cost	[49] [50] [73]
	Theoretical Cost	[68] [5] [52] [65] [73] [75] [74] [46] [78] [76]
Reliability	Downtime	[59] [56] [60] [7] [35] [58] [37]
	Failure Probability	[57]
	Recovery Probability	[56]
	Number of Recovered Components	[56] [67] [36] [77]
	Mean Time to Repair	[35]
	Mean Time Between Failures	[35]
Time Behavior	Application Delay	[59] [56] [7] [55] [41] [45] [44] [37] [36]
	Number of SLA Violations	[8] [30]
	Throughput	[59] [56] [7] [51] [72] [40] [39] [47] [42] [37]
	Execution Time	[27] [32] [63] [64] [65] [72] [73] [40] [54] [78] [77]
	Maintenance Batches	[32] [28]
	Maintenance Time	[59] [56] [7] [67] [8] [51] [62] [53] [71] [73] [6] [75] [61] [47] [44] [30]
	Migration Time	[7] [29] [8]
Resource Utilization	CPU/RAM/Disk Usage	[7] [27] [32] [63] [53] [6] [75] [78]
	Network Usage	[73] [40] [41] [39] [28] [66] [48] [78]
	Power Consumption	[57] [5] [73] [75] [42] [48] [37] [30] [77]

the modeling of specific scenarios, it usually undermines performance comparisons, especially as the source code of custom simulators is often private. Despite the predominance of custom simulators whose source code is not publicly available, some research works employed open-source simulation frameworks. In general, maintenance simulations use three types of simulation tools: (i) general-purpose simulators, (ii) network simulators, and (iii) maintenance-related simulators.

Malik et al. [66] modeled the coordination of patch distributions in software-defined vehicular networks using AnyLogic[14], a general-purpose simulation framework that provides a flexible programming interface supporting three simulation models: Agent-Based Simulation Modeling, Discrete Event Simulation Modeling, and System Dynamics Simulation Modeling. While some works employed general-purpose simulators such as AnyLogic aiming at flexible support for multiple types of conceptual models, others have chosen simulation tools with more specific purposes. Fakhrolmobasheri et al. [57] modeled a hypervisor rejuvenation scenario with the Möbius simulator[15], which focuses on modeling and analyzing stochastic processes such as Markov chains and Petri Nets. Möbius adopts a discrete-event simulation model with flexible monitoring capabilities that enable measuring several metrics (e.g., reliability, availability, performance, and security) at varied intervals (e.g., specific time points, over periods of time, or when the system reaches steady state) Torquato et al. [58] took a similar approach, modeling a hypervisor rejuvenation scenario with TimeNET[16], which focuses on providing flexible support to

[14]https://www.anylogic.com/
[15]https://www.mobius.illinois.edu/
[16]https://timenet.tu-ilmenau.de/#/

simulations based on Petri Nets (e.g., evaluation of models with nonexponentially distributed transition firing delays and support to Colored Stochastic Petri Nets).

As several maintenance-related activities involve intense network communication (e.g., patch distribution and application migrations), some research works employed network simulators during their evaluations. Banikhalaf et al. [41] considered a route repair scenario where the IoT infrastructure is composed of moving vehicles. The authors used NS-2 [93] alongside SUMO[17] to validate the proposed strategy. While NS-2 simulates various aspects of the wireless network infrastructure, such as routing and traffic control, SUMO enables modeling vehicle mobility based on real and synthetic traces. Nain et al. [40] [42] addressed a different scenario, employing the Cooja simulator[18] to perform network route updates in low-power and lossy IoT networks, where network devices and transmission modes are resource-constrained. While NS-2 addresses network simulation from a more general perspective (e.g., supporting both wired and wireless networks), Cooja focuses on simulating Wireless Sensor Networks (WSNs), providing built-in support for emulating real hardware platforms. In addition to supporting several network technologies, such as the Routing Protocol for Low Power and Lossy Networks, Cooja provides fine-grained control over network stacks and real-time monitoring of various metrics such as network throughput and power consumption.

Although general-purpose and network simulators provide suitable features for some maintenance scenarios, they lack support for modeling specific maintenance problems, highlighting the need for specialized simulators. Souza et al. [8] modeled an edge server maintenance scenario using EdgeSimPy [94], a simulation toolkit designed to ease the prototyping of resource management policies on edge infrastructures through system models that accurately represent various elements, including the lifecycle of containerized applications. While EdgeSimPy can model maintenance work involving multiple components (e.g., edge servers, applications, and network devices), other simulation toolkits concentrate on specific maintenance scenarios. An example of a specialized maintenance simulator is FUOTASim [95], used by Anastasiou et al. [73] to model firmware updates in IoT environments. In addition to simulating the behavior of wireless network protocols such as LoRaWAN[19], FUOTASim eases the monitoring of several metrics, such as patch distribution time and infrastructure power consumption.

4.5.3 Experimentation. Empirical experimentation consists of validating research prototypes in testbeds with characteristics similar to production environments, which enables the identification of real-world challenges and requirements. In addition to approximating research outcomes to practical applications, many research works present update/repair frameworks that rely on optimizations upon hardware-specific behaviors that are difficult to simulate accurately, which makes empirical experimentation even more attractive. In this context, one of the main challenges is related to the cost of building testbeds at scale, which involves the acquisition of several components (e.g., compute and network devices, cooling and power supply systems, among others) and continuous environment supervision. As a result, we observe that most existing academic efforts performed empirical tests on small-sized testbeds comprised of a few servers or personal computers equipped with networking and virtualization technologies [62] [54] [55] [64] [53] [67] [51] [61] [59] [56] [7] [52] [6] [75] [74] [44]. In contrast, some contributions included practical tests on mid-to-large infrastructures. For example, Leiba et al. [49] and Tapas et al. [50] used IoT testbeds composed of 48 and 50 single-board computers, respectively, while Russinovich et al. [60] evaluated their proposed server update approach by performing driver updates in Microsoft Azure's data centers with millions of servers.

4.5.4 Summary. Existing maintenance research efforts employ diverse validation approaches according to the addressed scenario, as shown in Table 7. In general, few maintenance strategies are validated through formal

[17]https://sourceforge.net/projects/sumo/
[18]https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja
[19]https://lora-alliance.org/about-lorawan/

modeling methods. This is mainly because maintenance scenarios often include non-deterministic phenomena caused by cascading events, which can be challenging to model and evaluate at scale. In contrast, several works use simulation tools to represent conceptual models with significant levels of detail at low implementation costs. Table 8 presents the list of simulators used to evaluate the proposed strategies. We can observe that most simulated evaluations use custom simulators, highlighting the absence of a *de facto* standard maintenance simulation toolkit. In addition, many existing research efforts perform empirical experimentation, as proposed solutions leverage hardware-specific behaviors that are difficult to model artificially.

Table 7. Summary of validation approaches used by the selected papers.

Validation Approaches	References
Simulation	[42] [40] [41] [73] [58] [66] [57] [8] [30] [48] [71] [68] [35] [65] [63] [5] [29] [28] [46] [45] [36] [78] [37] [32] [47] [39] [76] [77]
Experimentation	[32] [47] [39] [76] [77] [49] [52] [74] [50] [6] [75] [55] [51] [53] [54] [67] [61] [64] [62] [7] [59] [56] [60] [44] [72]
Formal Analysis	[72] [27]

Table 8. Summary of simulation tools used to validate the strategies proposed by the selected papers.

Simulators	References
Custom	[68] [32] [5] [35] [29] [63] [71] [45] [28] [46] [47] [36] [78] [37]
Möbius	[57]
EdgeSimPy	[8] [30]
Cooja	[40] [42]
FUOTASim	[73]
NS-2	[41]
MATLAB	[39]
TimeNET	[58]
AnyLogic	[66]
ds-sim	[48]

5 FUTURE RESEARCH DIRECTIONS

Although existing efforts have covered several challenges related to maintenance aiming at cloud, edge, and IoT environments, we observe that some research topics require further investigation. This section outlines our vision for the research landscape in this area, correlating the existing work (presented in the previous section) with the identified challenges. Figure 10 introduces a block diagram that gives a high-level perspective on the addressed research challenges and the missing pieces that represent opportunities for future research endeavors.

Our organization of the research landscape categorizes challenges into three layers: (i) infrastructure-related challenges, (ii) platform-related challenges, and (iii) application-related challenges. Overall, the research gaps related to the infrastructure revolve around ensuring the sustainability of maintenance policies, given that maintenance work is typically highly demanding on computing resources (§5.5), and the ability to handle specific workload characteristics and infrastructure layout (§5.6). On the other hand, the research gaps related to the platform converge to the need to develop new maintenance policies optimized to handle the specific features and requirements of containerization, which has emerged as a leading virtualization technology (§5.3), and to minimize issues related to resource contention and virtualization overheads (§5.4). Finally, the research gaps related to applications are focused on designing maintenance policies that make allocation decisions considering the needs of modern software architectures (§5.1) and QoS requirements (§5.2). The remainder of this section discusses the identified gaps and sheds light on future research directions.

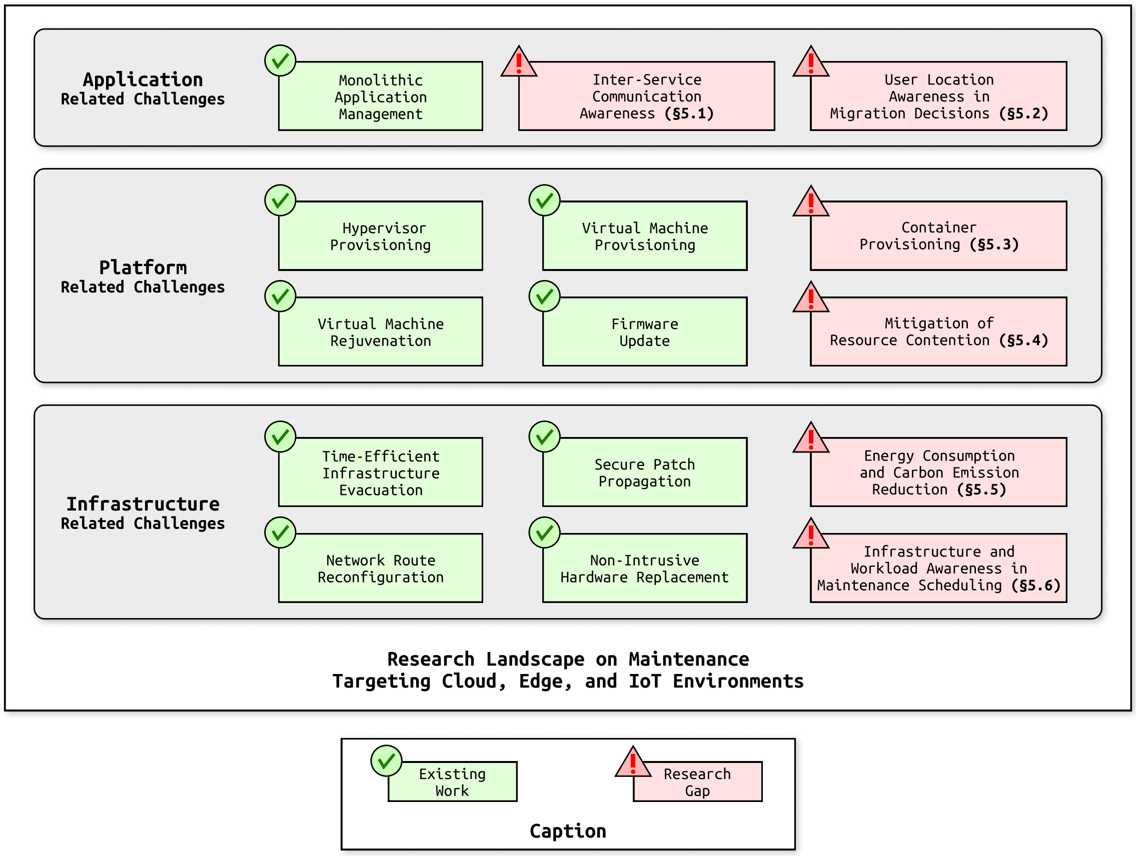
Application Related Challenges
Monolithic Application Management
Inter-Service Communication Awareness (§5.1)
User Location Awareness in Migration Decisions (§5.2)
Platform Related Challenges
Hypervisor Provisioning
Virtual Machine Provisioning
Container Provisioning (§5.3)
Virtual Machine Rejuvenation
Firmware Update
Mitigation of Resource Contention (§5.4)
Infrastructure Related Challenges
Time-Efficient Infrastructure Evacuation
Secure Patch Propagation
Energy Consumption and Carbon Emission Reduction (§5.5)
Network Route Reconfiguration
Non-Intrusive Hardware Replacement
Infrastructure and Workload Awareness in Maintenance Scheduling (§5.6)
Research Landscape on Maintenance Targeting Cloud, Edge, and IoT Environments
Existing Work
Research Gap
Caption

Fig. 10. Overview of research challenges related to maintenance targeting cloud, edge, and IoT infrastructures and missing pieces that represent opportunities for future research endeavors.

5.1 Inter-Service Communication Awareness

Application composition architectures have been attracting growing interest within the IT community by facilitating the implementation of more sophisticated resource management strategies than traditional monoliths [96]. Demand variations in specific components of monolithic applications are usually hard to leverage as the whole application is deployed as a single software unit. Conversely, services from composite applications are deployed independently, allowing infrastructure operators to enhance resource utilization by assigning more resources for services under intense demand and releasing resources from the less occupied ones. In addition, the loosely coupled model employed by composite applications improves software maintainability, as changes in specific services do not affect others.

Whereas application composition widens the allocation options for IT operators, its distributed nature introduces some challenges that resource management decisions must consider during maintenance work. In particular, services from composite applications must be provisioned near each other to avoid excessive latency increases. Also, the absence of strategic service positioning can lead to loopy communication patterns that might saturate the network, especially in the case of composite applications with complex data flows. Although a few initiatives tackle the maintenance of infrastructures hosting composite applications [63], proposed solutions are not focused on optimizing the application provisioning decisions. Therefore, more sophisticated maintenance strategies considering the performance requirements of composite applications are needed to alleviate the impact of maintenance work on end-users' quality of service.

5.2 User Location Awareness in Migration Decisions

Many maintenance activities require temporarily disabling newly-updated components for changes to take effect [32] [59]. As applications increasingly depend on high availability and maintenance-led downtime can take significant time, most solutions resort to migration techniques that relocate applications among available hosts to alleviate the maintenance side effects [56] [7] [29]. During maintenance in cloud data centers, applications from servers waiting for an update can be evacuated to any alternative hosts with sufficient available capacity, regardless of their location within the data center. Such provisioning decision is allowed as cloud servers are usually homogeneous in terms of processing power, and data center networks comprise high-speed network switches with spare components that reduce the chances of communication bottlenecks.

While location awareness is not a primary concern for migrations during maintenance on cloud data centers, some attributes reverse the situation on edge and IoT infrastructures. First, edge and IoT devices often display variable performance capabilities due to their heterogeneous hardware configurations, limiting the provisioning options. Second, inter-device communication in such environments is provided by less robust (sometimes shared) networks, which are more susceptible to instability during demanding periods. Although a few maintenance solutions incorporate location awareness into migration decisions [28] [8], they assume sequential migrations, overlooking the required coordination of concurrent migrations within the network. In addition, they lack migration coordination methods to optimize the network communication of composite applications, which can experience exponential latency increases due to loopy communication paths (see more details in §5.1). Therefore, further investigation is necessary to effectively perform location-aware migration decisions during maintenance on edge and IoT sites and avoid potential application performance drops during such demanding operations.

5.3 Optimized Provisioning of Containerized Applications

Modern computing platforms have been following a trend toward adopting containers as the preferred virtualization technology over VMs [9]. One of the primary motivations for this shift relies on the differences between the image filesystems of VMs and containers. While VM images are monolithic, most container solutions employ layered filesystems that divide images into a top writable layer that stores any container-specific changes and multiple underlying read-only layers carrying the other software instructions (e.g., build packs, libraries, and other dependencies). While co-hosted VMs have a full copy of their base images regardless of repeated software instructions, co-hosted containers can share common layers. Although these architectural differences might look subtle at a glance, they grant containers considerably faster provisioning time and lower disk footprint than VMs.

Despite the undisputed isolation provided by VMs, containers have been gaining significant attention by enabling improved resource usage and greater flexibility for large-scale application deployments. Although virtualization techniques have been extensively exploited by maintenance strategies [27] [58] [56] [29], the majority of existing approaches are designed for the VM model—so they cannot leverage the full potential of containerization. Olorunnife et al. [61] was the only analyzed work that considered containerized deployments. However, their proposed solution did not perform any specific decision-making based on container characteristics. Also, the authors did not mention container orchestration platforms such as Kubernetes[20], which already implement several features to support maintenance operations, including commands to drain nodes during rolling upgrades[21] and mark them based on custom maintenance logic[22].

It is worth noting that Kubernetes natively monitors and corrects the state of certain components according to predefined desired specifications (so-called Desired versus Actual State). Nevertheless, as a general-purpose orchestration platform, Kubernetes lacks advanced maintenance policies with optimized rules to, e.g., define the order in which nodes are updated, which optimized targets are prioritized during node draining procedures, and how container image updates are pulled from container registries. Based on that, further research is necessary to close the gap between state-of-the-art maintenance approaches and industry-standard platforms like Kubernetes.

5.4 Mitigation of Resource Contention

Virtualization sits at the core of most modern computing platforms, providing increased provisioning flexibility and fine-grained control of physical resources [9]. One of the main features provided by virtualization is multiplexing, which allows multiple applications to run on top of the same physical resources. Whereas multiplexing widens the provisioning options, it introduces performance concerns related to resource contention (also known as performance interference), where co-located applications, especially those that rely on the same type of resource (e.g., CPU cache and I/O), degrade each other performance.

Existing research works have employed virtualization techniques for optimizing several provisioning decisions during maintenance. A typical example resorts to migrating applications across the infrastructure when applying updates that require temporarily disabling the affected components [56] [7] [29] [8]. Although the existing solutions successfully mitigate some undesirable events such as application downtime and network bottlenecks, they overlook the potential performance degradation caused by stacking multiple applications on the same host. Therefore, there is a need for further study of interference-aware allocation approaches that cope with the requirements of maintenance scenarios to mitigate the application performance losses experienced during maintenance work.

5.5 Energy Consumption and Carbon Emission Reduction

Despite the existing endeavors towards power consumption reductions during maintenance, existing solutions predominantly focus on IoT environments, acting under the motivation that IoT nodes have limited power supply [63] [40] [75]. Consequently, proposed solutions are not primarily concerned with ensuring that maintenance work meets sustainability goals, especially on cloud and edge infrastructures. Also, incorporating energy and carbon emission awareness during maintenance includes the design of maintenance strategies capable of leveraging renewable energy sources (e.g., wind and solar), which implies handling the unstable and dynamic behavior of most renewable energy sources—for instance, wind energy is influenced by wind speed, and solar energy varies according to temperature and radiation [97].

Many research efforts present dynamic approaches that switch between conventional fossil fuels and renewable energy based on various criteria, such as variations in weather conditions and application workload [98] [99]. Nevertheless, existing solutions are not optimized for the particularities of maintenance work, which introduces specific provisioning constraints (e.g., when applications cannot be provisioned on outdated components) and strict deadlines (e.g., during security patching). Accordingly, new energy and carbon-aware maintenance strategies must be developed to ensure maintenance work is as sustainable as possible. Although this concern applies to

[20]https://kubernetes.io/
[21]https://kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/
[22]https://kubernetes.io/docs/concepts/architecture/nodes/#manual-node-administration

edge and IoT infrastructures, it might be mainly directed to cloud data centers, which can consume considerable energy during resource-intensive activities such as maintenance.

5.6 Infrastructure and Workload Awareness in Maintenance Scheduling

Prioritization policies play a critical role during the maintenance of large-scale infrastructures, where the order in which events occur affects the quality of maintenance work. In such a scenario, optimal scheduling algorithms may define the order of several decisions (e.g., component updates and application migrations) based on various factors, such as the criticality of affected components within the infrastructure and the application workload. For instance, during critical security updates, servers from sandbox environments can receive lower priority than production servers, as sandbox servers can be temporarily disabled without affecting critical applications, while postponing the update of production servers can facilitate vulnerability propagation. In addition, defining a proper schedule for distributing and applying updates can also influence the effectiveness of maintenance work, especially when multiple updates are released within a short period. Whereas some updates might be scheduled for low-demand periods, postponing others could severely affect the infrastructure's security and performance.

Existing maintenance efforts employ prioritization policies to support the schedule of distributing patches [40], migrating applications [29] [58], and updating components [27] [32] [8]. However, they predominantly concentrate on generic goals such as reducing maintenance time, which does not necessarily imply a proper schedule for maintenance decisions. New robust prioritization policies considering other parameters such as infrastructure organization (e.g., sandbox and production environments) and workload characteristics (e.g., how many users and applications depend on affected components) must be developed to improve the effectiveness of maintenance work. As most maintenance scenarios require the efficient coordination of multiple objectives, new research endeavors in the field should consider using dynamic techniques such as ML and RL to achieve cost-efficient solutions, which, despite the promising advancements, are still barely explored by maintenance approaches, as shown in Section 4.3.4. Another promising way is to design novel metrics that characterize the impact of prioritization policies on the quality of maintenance decisions.

6 CONCLUDING REMARKS

The ever increasing demand for connectivity has boosted the advancement of the Internet of Things, which embeds sensing and networking capabilities in ordinary objects. One of the consequences of such movement was the rise of new classes of applications whose latency and bandwidth requirements conflict with the traditional computing model based on cloud data centers, which, despite their powerful computing capabilities, are limited by their distance to data sources. In response to such a challenge, the IT community promoted a shift towards the cooperation of three paradigms: Internet of Things, Edge Computing, and Cloud Computing. On one end, IoT devices deliver computing intelligence for daily tasks. In the middle, edge servers provide real-time feedback to latency-sensitive IoT applications. On the other end, cloud data centers offer abundant processing power to handle resource-intensive tasks and to provide long-term storage.

Despite the promising use cases for the unified IoT-Edge-Cloud model, IT personnel face the challenge of implementing efficient maintenance strategies to preserve the environment's performance and security, which resorts to understanding the numerous requirements, demands, and challenges of such a complex ecosystem. While the abundant resources from the academic literature could help in that regard, extracting actionable insights from their findings and drawing parallels between multiple works is challenging due to the massive number of existing papers. In this context, there is a need for review papers providing an overview of existing solutions and indications of open challenges to lower the barrier to entry for new researchers.

This paper presented a comprehensive survey of maintenance research aimed at cloud, edge, and IoT environments. As the scope of our study intersected multiple research subjects, we proposed a taxonomy that provides a logical organization of the existing works based on various characteristics, such as maintenance approaches, employed techniques, and metrics of interest. In addition, we shed light on challenges and opportunities that researchers and practitioners must address to advance the field. We believe this paper will play a crucial role in facilitating the understanding and motivating further research on maintenance in the addressed paradigms.

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 (LAD-IDEIA/PUCRS) for providing support and technological resources for this project.

REFERENCES

[1] Peter Mell and Timothy Grance. The nist definition of cloud computing, 2011-09-28 2011.

[2] Rajkumar Buyya, Satish Narayana Srirama, Giuliano Casale, Rodrigo Calheiros, Yogesh Simmhan, Blesson Varghese, Erol Gelenbe, Bahman Javadi, Luis Miguel Vaquero, Marco AS Netto, et al. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM computing surveys*, 51(5):1–38, 2018.

[3] Rosilah Hassan, Faizan Qamar, Mohammad Kamrul Hasan, Azana Hafizah Mohd Aman, and Amjed Sid Ahmed. Internet of things and its applications: A comprehensive survey. *Symmetry*, 12(10):1674, 2020.

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

[5] Weiwei Wu, Jianping Wang, Kejie Lu, Wen Qi, Feng Shan, and Junzhou Luo. Providing service continuity in clouds under power outage. *IEEE Transactions on Services Computing*, 13(5):930–943, 2017.

[6] Yan-Hong Fan, Mei-Qin Wang, Yan-Bin Li, Kai Hu, and Mu-Zhou Li. A secure iot firmware update scheme against scpa and dos attacks. *Journal of Computer Science and Technology*, 36(2):419–433, 2021.

[7] Tu Dinh Ngoc, Boris Teabe, Alain Tchana, Gilles Muller, and Daniel Hagimont. Mitigating vulnerability windows with hypervisor transplant. In *European Conference on Computer Systems*, pages 162–177, 2021.

[8] Paulo S Souza, Tiago C Ferreto, Fábio D Rossi, and Rodrigo N Calheiros. Location-aware maintenance strategies for edge computing infrastructures. *IEEE Communications Letters*, 26(4):848–852, 2022.

[9] Yaser Mansouri and M Ali Babar. A review of edge computing: Features and resource virtualization. *Journal of Parallel and Distributed Computing*, 150:155–183, 2021.

[10] Parvaneh Asghari, Amir Masoud Rahmani, and Hamid Haj Seyyed Javadi. Internet of things applications: A systematic review. *Computer Networks*, 148:241–261, 2019.

[11] Qi Jing, Athanasios V Vasilakos, Jiafu Wan, Jingwei Lu, and Dechao Qiu. Security of the internet of things: Perspectives and challenges. *Wireless Networks*, 20(8):2481–2501, 2014.

[12] Burak Kantarci and Hussein T Mouftah. Sensing services in cloud-centric internet of things: A survey, taxonomy and challenges. In *International Conference on Communication Workshop*, pages 1865–1870, London, 2015. IEEE.

[13] Gary Josebeck and Arun Gowtham. Demystifying the pf curve & augmenting machine learning for maintenance optimization. In *2022 Annual Reliability and Maintainability Symposium*, pages 1–5. IEEE, 2022.

[14] Ashok Prajapati, James Bechtel, and Subramaniam Ganesan. Condition based maintenance: a survey. *Journal of Quality in Maintenance Engineering*, 18(4):384–400, 2012.

[15] Ruben Sipos, Dmitriy Fradkin, Fabian Moerchen, and Zhuang Wang. Log-based predictive maintenance. In *International Conference on Knowledge Discovery and Data Mining*, pages 1867–1876, New York, 2014. ACM.

[16] Donghwan Kim, Seungchul Lee, and Daeyoung Kim. An applicable predictive maintenance framework for the absence of run-to-failure data. *Applied Sciences*, 11(11):5180, 2021.

[17] Luca Silvestri, Antonio Forcina, Vito Introna, Annalisa Santolamazza, and Vittorio Cesarotti. Maintenance transformation through industry 4.0 technologies: A systematic literature review. *Computers in Industry*, 123, 2020.

[18] Hans Christian Benestad, Bente Anda, and Erik Arisholm. Understanding software maintenance and evolution by analyzing individual changes: a literature review. *Journal of Software Maintenance and Evolution: Research and Practice*, 21(6):349–378, 2009.

[19] Martin Monperrus. Automatic software repair: A bibliography. *ACM Computing Surveys*, 51(1):1–24, 2018.

[20] Justyna Petke, Saemundur O Haraldsson, Mark Harman, William B Langdon, David R White, and John R Woodward. Genetic improvement of software: a comprehensive survey. *IEEE Transactions on Evolutionary Computation*, 22(3):415–432, 2017.

[21] Mostafa Fadaeefath Abadi, Fariborz Haghighat, and Fuzhan Nasiri. Data center maintenance: applications and future research directions. *Facilities*, 38(9/10):691–714, 2020.

[22] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & information systems engineering*, 6:239–242, 2014.

[23] Tiago Zonta, Cristiano André Da Costa, Rodrigo da Rosa Righi, Miromar Jose de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. Predictive maintenance in the industry 4.0: A systematic literature review. *Computers & Industrial Engineering*, 150:106889, 2020.

[24] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys*, 51(4):1–33, 2018.

[25] Xunyun Liu and Rajkumar Buyya. Resource management and scheduling in distributed stream processing systems: A taxonomy, review, and future directions. *ACM Computing Surveys*, 53(3):1–41, 2020.

[26] Raquel V Lopes and Daniel Menascé. A taxonomy of job scheduling on distributed computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(12):3412–3428, 2016.

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

[28] Weigang Hou, Wenxiao Li, Lei Guo, Yiwei Sun, and Xintong Cai. Recycling edge devices in sustainable internet of things networks. *Internet of Things Journal*, 4(5):1696–1706, 2017.

[29] Chen Ying, Baochun Li, Xiaodi Ke, and Lei Guo. Raven: Scheduling virtual machine migration during datacenter upgrades with reinforcement learning. *Mobile Networks and Applications*, 27(1):1–12, 2022.

[30] Felipe Rubin, Paulo Souza, and Tiago Ferreto. Reducing power consumption during server maintenance on edge computing infrastructures. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, pages 691–698, 2023.

[31] Kathryn A Dowsland and William B Dowsland. Packing problems. *European journal of operational research*, 56(1):2–14, 1992.

[32] Long Wang, Harigovind V Ramasamy, and Richard E Harper. Scheduling physical machine maintenance on qualified clouds: What if migration is not allowed? In *International Conference on Cloud Computing*, pages 485–492. IEEE, 2020.

[33] Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News*, 32(1):36–52, 2001.

[34] Arjang A Assad. Multicommodity network flows—a survey. *Networks*, 8(1):37–91, 1978.

[35] Deepika Saxena and Ashutosh Kumar Singh. Ofp-tm: an online vm failure prediction and tolerance model towards high availability of cloud computing environments. *The Journal of Supercomputing*, 78(6):8003–8024, 2022.

[36] Ziyuan Wang, Zekai Zhang, Jingjing Wang, Chunxiao Jiang, Wei Wei, and Yong Ren. Auv-assisted node repair for iout relying on multiagent reinforcement learning. *IEEE Internet of Things Journal*, 11(3):4139–4151, 2024.

[37] Yuting Wang, Xiaofan Han, and Shunfu Jin. Performance analysis of a vm-pm repair strategy in mec-enabled wireless systems with bursty traffic. *IEEE Transactions on Vehicular Technology*, 73(1):1146–1161, 2024.

[38] Jacob H Cox, Joaquin Chung, Sean Donovan, Jared Ivey, Russell J Clark, George Riley, and Henry L Owen. Advancing software-defined networks: A survey. *IEEE Access*, 5:25487–25526, 2017.

[39] Wei Ren, Yan Sun, Hong Luo, and Mohsen Guizani. Bllc: A batch-level update mechanism with low cost for sdn-iot networks. *IEEE Internet of Things Journal*, 6(1):1210–1222, 2018.

[40] Zulqar Nain, Arslan Musaddiq, Yazdan Ahmad Qadri, Ali Nauman, Muhammad Khalil Afzal, and Sung Won Kim. Riata: A reinforcement learning-based intelligent routing update scheme for future generation iot networks. *IEEE Access*, 9:81161–81172, 2021.

[41] Mustafa Banikhalaf, Ahmad M Manasrah, Ahmed F AlEroud, Nabhan Hamadneh, Ahmad Qawasmeh, and Ahmed Y Al-Dubai. A reliable route repairing scheme for internet of vehicles. *International Journal of Computer Applications in Technology*, 61(3):229–238, 2019.

[42] Zulqar Nain, Arslan Musaddiq, Yazdan Ahmad Qadri, and Sung Won Kim. History-aware adaptive route update scheme for low-power and lossy networks. In *International Conference on Information and Communication Technology Convergence*, pages 1830–1834. IEEE, 2021.

[43] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[44] Muhammad Taqi Raza, Zhowei Tan, Ali Tufail, and Fatima Muhammad Anwar. Lte nfv rollback recovery. *IEEE Transactions on Network and Service Management*, 19(3):2468–2477, 2022.

[45] Huawei Huang and Song Guo. Proactive failure recovery for nfv in distributed edge computing. *IEEE Communications Magazine*, 57(5):131–137, 2019.

[46] Zhenyi Huang and Huawei Huang. Proactive failure recovery for stateful nfv. In *International Conference on Parallel and Distributed Systems*, pages 536–543. IEEE, 2020.

[47] Yu Wu, Duo Liu, Yujuan Tan, Moming Duan, Longpan Luo, Weilve Wang, and Xianzhang Chen. LFPR: A lazy fast predictive repair strategy for mobile distributed erasure coded cluster. *IEEE Internet of Things Journal*, 10(1):704–719, 2023.

[48] Rekha Nachiappan, Rodrigo N. Calheiros, Kenan M. Matawie, and Bahman Javadi. Optimized proactive recovery in erasure-coded cloud storage systems. *IEEE Access*, 11(1):38226–38239, 2023.

[49] Oded Leiba, Ron Bitton, Yechiav Yitzchak, Asaf Nadler, Davidoz Kashi, and Asaf Shabtai. Iotpatchpool: Incentivized delivery network of iot software updates based on proofs-of-distribution. *Pervasive and Mobile Computing*, 58:1–21, 2019.

[50] Nachiket Tapas, Yechiav Yitzchak, Francesco Longo, Antonio Puliafito, and Asaf Shabtai. P4uiot: Pay-per-piece patch update delivery for iot using gradual release. *Sensors*, 20(7):1–27, 2020.

[51] Elizabeth Nathania Witanto, Yustus Eko Oktian, Sang-Gon Lee, and Jin-Heung Lee. A blockchain-based ocf firmware update for iot devices. *Applied Sciences*, 10(19):1–22, 2020.

[52] Tatsuhiro Fukuda and Kazumasa Omote. Efficient blockchain-based iot firmware update considering distribution incentives. In *Conference on Dependable and Secure Computing*, pages 1–8. IEEE, 2021.

[53] Antonio Langiu, Carlo Alberto Boano, Markus Schuß, and Kay Römer. Upkit: An open-source, portable, and lightweight update framework for constrained iot devices. In *International Conference on Distributed Computing Systems*, pages 2101–2112. IEEE, 2019.

[54] N Asokan, Thomas Nyman, Norrathep Rattanavipanon, Ahmad-Reza Sadeghi, and Gene Tsudik. Assured: Architecture for secure software update of realistic embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2290–2300, 2018.

[55] Xinchi He, Sarra Alqahtani, Rose Gamble, and Mauricio Papa. Securing over-the-air iot firmware updates using blockchain. In *International Conference on Omni-Layer Intelligent Systems*, pages 164–171, 2019.

[56] Frederico Cerveira, Raul Barbosa, and Henrique Madeira. Mitigating virtualization failures through migration to a co-located hypervisor. *IEEE Access*, 9:105255–105269, 2021.

[57] Sharifeh Fakhrolmobasheri, Ehsan Ataie, and Ali Movaghar. Modeling and evaluation of power-aware software rejuvenation in cloud systems. *Algorithms*, 11(10):160, 2018.

[58] Matheus Torquato, Paulo Maciel, and Marco Vieira. A model for availability and security risk evaluation for systems with vmm rejuvenation enabled by vm migration scheduling. *IEEE Access*, 7:138315–138326, 2019.

[59] Andrea Segalini, Dino Lopez Pacheco, Guillaume Urvoy-Keller, Fabien Hermenier, and Quentin Jacquemart. Hy-fix: Fast in-place upgrades of kvm hypervisors. *IEEE Transactions on Cloud Computing*, 10(4):2679–2690, 2022.

[60] Mark Russinovich, Naga Govindaraju, Melur Raghuraman, David Hepkin, Jamie Schwartz, and Arun Kishan. Virtual machine preserving host updates for zero day patching in public cloud. In *European Conference on Computer Systems*, pages 114–129, 2021.

[61] Kolade Olorunnife, Kevin Lee, and Jonathan Kua. Automatic failure recovery for container-based iot edge applications. *Electronics*, 10(23):3047, 2021.

[62] Martin Weißbach, Nguonly Taing, Markus Wutzler, Thomas Springer, Alexander Schill, and Siobhan Clarke. Decentralized coordination of dynamic software updates in the internet of things. In *World Forum on Internet of Things*, pages 171–176. IEEE, 2016.

[63] Ngoc Hai Bui, Kim Khoa Nguyen, Chuan Pham, and Mohamed Cheriet. Energy efficient software update mechanism for networked iot devices. In *Global Communications Conference*, pages 1–6. IEEE, 2019.

[64] Hemant Gupta and Paul C Van Oorschot. Onboarding and software update architecture for iot devices. In *International Conference on Privacy, Security and Trust*, pages 1–11. IEEE, 2019.

[65] Mohammad Salar Arbabi and Mehdi Shajari. Decentralized and secure delivery network of iot update files based on ethereum smart contracts and blockchain technology. In *Annual International Conference on Computer Science and Software Engineering*, pages 110–119. ACM, 2019.

[66] Asad Waqar Malik, Anis U. Rahman, Arsalan Ahmad, and Max Mauro Dias Santos. Over-the-air software-defined vehicle updates using federated fog environment. *IEEE Transactions on Network and Service Management*, 19(4):5078–5089, 2022.

[67] Jingzhu He, Ting Dai, Xiaohui Gu, and Guoliang Jin. Hangfix: automatically fixing software hang bugs for production cloud systems. In *ACM Symposium on Cloud Computing*, pages 344–357. ACM, 2020.

[68] Haining Meng, Xu Zhang, Lei Zhu, Lei Wang, and Zijiang Yang. Optimizing software rejuvenation policy based on cdm for cloud system. In *Conference on Industrial Electronics and Applications*, pages 1850–1854. IEEE, 2017.

[69] Oleksandr Rolik, Sergii Telenyk, and Eduard Zharikov. Management of services of a hyperconverged infrastructure using the coordinator. In *International Conference on Computer Science, Engineering and Education Applications*, pages 456–467. Springer, 2018.

[70] Sam Halabi. *Hyperconverged Infrastructure Data Centers: Demystifying HCI*. Cisco Press, 2019.

[71] Shin-Ming Cheng, Pin-Yu Chen, Ching-Chao Lin, and Hsu-Chun Hsiao. Traffic-aware patching for cyber security in mobile iot. *IEEE Communications Magazine*, 55(7):29–35, 2017.

[72] Jen-Wei Hu, Lo-Yao Yeh, Shih-Wei Liao, and Chu-Sing Yang. Autonomous and malware-proof blockchain-based firmware update platform with efficient batch verification for internet of things devices. *Computers & Security*, 86:238–252, 2019.

[73] A Anastasiou, Panayiotis Christodoulou, Klitos Christodoulou, Vasos Vassiliou, and Zinon Zinonos. Iot device firmware update over lora: The blockchain solution. In *International Conference on Distributed Computing in Sensor Systems*, pages 404–411. IEEE, 2020.

[74] Woei-Jiunn Tsaur, Jen-Chun Chang, and Chin-Ling Chen. A highly secure iot firmware update mechanism using blockchain. *Sensors*, 22(2):530, 2022.

[75] Njabulo Sakhile Mtetwa, Paul Tarwireyi, Cecilia Nombuso Sibeko, Adnan Abu-Mahfouz, and Matthew Adigun. Blockchain-based security model for lorawan firmware updates. *Journal of Sensor and Actuator Networks*, 11(1):5, 2022.

[76] Gabriel Solomon, Peng Zhang, Rachael Brooks, and Yuhong Liu. A secure and cost-efficient blockchain facilitated iot software update framework. *IEEE Access*, 11:44879–44894, 2023.

[77] Songran Liu, Mingsong Lv, Wei Zhang, Xu Jiang, Chuancai Gu, Tao Yang, Wang Yi, and Nan Guan. Light flash write for efficient firmware update on energy-harvesting iot devices. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.
[78] Haruto Taka, Fujun He, and Eiji Oki. Joint service placement and user assignment model in multi-access edge computing networks against base-station failure. *International Journal of Network Management*, page e2233, 2023.
[79] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Georges Kaddoum, Albert Y Zomaya, and Rajiv Ranjan. A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. *IEEE Transactions on Network and Service Management*, 16(3):924–935, 2019.
[80] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *International Symposium on Fault-Tolerant Computing*, pages 381–390. IEEE, 1995.
[81] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
[82] William H Sanders and John F Meyer. Stochastic activity networks: formal definitions and concepts. In *School organized by the European Educational Forum*, pages 315–343. Springer, 2000.
[83] Gianfranco Ciardo and Kishor S Trivedi. A decomposition approach for stochastic reward net models. *Performance Evaluation*, 18(1):37–59, 1993.
[84] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer set solving in practice*. Springer Nature, 2022.
[85] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.
[86] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14:241–258, 2020.
[87] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
[88] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: Past, present, and future. *Multimedia tools and applications*, 80:8091–8126, 2021.
[89] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
[90] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.
[91] Maya Dotan, Yvonne-Anne Pignolet, Stefan Schmid, Saar Tochner, and Aviv Zohar. Survey on blockchain networking: Context, state-of-the-art, challenges. *ACM Computing Surveys*, 54(5):1–34, 2021.
[92] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019.
[93] Eitan Altman and Tania Jimenez. Ns simulator for beginners. *Synthesis Lectures on Communication Networks*, 5(1):1–184, 2012.
[94] Paulo S. Souza, Tiago Ferreto, and Rodrigo N. Calheiros. EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies. *Future Generation Computer Systems*, 148:446–459, 2023.
[95] Khaled Abdelfadeel, Tom Farrell, David McDonald, and Dirk Pesch. How to make firmware updates over lorawan possible. In *International Symposium on World of Wireless Mobile and Multimedia Networks*, pages 16–25. IEEE, 2020.
[96] David S Linthicum. Practical use of microservices in moving workloads to the cloud. *IEEE Cloud Computing*, 3(5):6–9, 2016.
[97] Daming Zhao and Jiantao Zhou. An energy and carbon-aware algorithm for renewable energy usage maximization in distributed cloud data centers. *Journal of Parallel and Distributed Computing*, 165:156–166, 2022.
[98] Ting Yang, Yucheng Hou, Young Choon Lee, Hao Ji, and Albert Y Zomaya. Power control framework for green data centers. *IEEE Transactions on Cloud Computing*, 10(4):2876–2886, 2020.
[99] Ouzhu Han, Tao Ding, Xiaosheng Zhang, Chenggang Mu, Xinran He, Hongji Zhang, Wenhao Jia, and Zhoujun Ma. A shared energy storage business model for data center clusters considering renewable energy uncertainties. *Renewable Energy*, 202:1273–1290, 2023.