applied sciences

MDPI

Systematic Review

On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study

Vincent Bushong [1,*], Amr S. Abdelfattah [1], Abdullah A. Maruf [1], Dipta Das [1], Austin Lehman [1], Eric Jaroszewski [1], Michael Coffey [1], Tomas Cerny [1,*], Karel Frajtak [2], Pavel Tisnovsky [3] and Miroslav Bures [2]

1 Computer Science, Baylor University, One Bear Place #97141, Waco, TX 76798, USA; amr_elsayed1@baylor.edu (A.S.A.); Maruf_Maruf1@baylor.edu (A.A.M.); dipta_das1@baylor.edu (D.D.); austin_lehman1@baylor.edu (A.L.); eric_jaroszewski1@baylor.edu (E.J.); Michael_Coffey@baylor.edu (M.C.)
2 Computer Science, Faculty of Electrical Engineering, Czech Technical University, Karlovo Nam. 13, 121 35 Prague, Czech Republic; frajtak@fel.cvut.cz (K.F.); buresm3@fel.cvut.cz (M.B.)
3 Red Hat, Purkynova 99, 612 00 Brno, Czech Republic; ptisnovs@redhat.com
* Correspondence: vincent_bushong1@baylor.edu (V.B.); tomas_cerny@baylor.edu (T.C.)

Abstract: Microservice architecture has become the leading design for cloud-native systems. The highly decentralized approach to software development consists of relatively independent services, which provides benefits such as faster deployment cycles, better scalability, and good separation of concerns among services. With this new architecture, one can naturally expect a broad range of advancements and simplifications over legacy systems. However, microservice system design remains challenging, as it is still difficult for engineers to understand the system module boundaries. Thus, understanding and explaining the microservice systems might not be as easy as initially thought. This study aims to classify recently published approaches and techniques to analyze microservice systems. It also looks at the evolutionary perspective of such systems and their analysis. Furthermore, the identified approaches target various challenges and goals, which this study analyzed. Thus, it provides the reader with a roadmap to the discipline, tools, techniques, and open challenges for future work. It provides a guide towards choices when aiming for analyzing cloud-native systems. The results indicate five analytical approaches commonly used in the literature, possibly in combination, towards problems classified into seven categories.

Keywords: microservices; system analysis; architectural degradation; software architecture reconstruction

Citation: Bushong, V.; Abdelfattah, A.S.; Maruf, A.A.; Das, D.; Lehman, A.; Jaroszewski, E.; Coffey, M.; Cerny, T.; Frajtak, K.; Tisnovsky, P.; et al. On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study. *Appl. Sci.* **2021**, *11*, 7856. https://doi.org/10.3390/app11177856

Academic Editor: Sofie Van Hoecke

Received: 22 July 2021
Accepted: 24 August 2021
Published: 26 August 2021

1. Introduction

Cloud-native systems take full advantage of distributed computing offered by the cloud delivery model. These systems are fueled mainly by microservice architecture [1], an architectural style where the system is broken down into reusable, lightweight, and granular services that interact with one another. In industry, microservices are rapidly gaining popularity, with companies such as Amazon, Netflix, and Spotify acting as role models in this trend. The increasing popularity of microservices can be attributed to the fact that they are easy to scale and more resilient to faults than other architectural styles when implemented properly [2]. Microservices are independently deployed, have lower coupling, and are self-contained. This means that services can be developed and tested by different development teams, dramatically lowering development time. It is expected that microservices become easier to maintain due to their smaller size and scope. However, is this really the case from a global perspective?

Since development teams are given more freedom for microservice evolution, a mechanism informing other teams about what has changed or evolved is missing. Thus, microservice architecture is not without its drawbacks. With the highly decentralized development and design of microservices, it becomes difficult to maintain a centralized reference of the architectural design. As a result, the system becomes vulnerable to architectural degrada-

tion, a phenomenon where changes to the codebase cause the architecture to deviate from the originally intended design.

One answer to face these challenges (centralized architectural design and architectural degradation) is to extract information about microservices to better explain them. This is accomplished by system analysis. However, what is the recent practice used in the literature to perform such analysis? We could use static or dynamic analysis on conventional systems, but how do these differ for microservices, and do they pose any limitations? To deal with these challenges more effectively, we should know about the recent microservice-specific practice.

The goal of this study is to report on the practice of analyzing microservice architecture. There are many questions about microservices that must be answered, such as finding most efficient and accurate methods for analyzing and debugging microservice systems. The unique characteristics of microservices make analysis simultaneously challenging and valuable. Thus, this study also focuses on the issues of how microservice architecture evolves and possibly degrades over time. In particular, this study considers:

- challenges in the analysis of the system architecture;
- fault detection, prevention, and root cause analysis of failure states;
- the practice of migration of monolithic systems into microservices;
- analysis role in system evolution, technical debt, and architecture degradation;
- how quality aspects are analyzed in microservices,
- the relationship between microservices and other architectures.

So far, no other recent study has been conducted to organize the topic of microservice analysis. Thus, we compile a mapping study of the most relevant and recent microservice analysis research from six scientific databases in the last 3 years.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 outlines the research methods we used to collect and analyze the current literature, including the definition of our research questions. In Section 4, we discuss our analysis of the works we identified. Section 5 contains our discussion of the results and the answers to our research questions. Section 6 discusses threats to validity. Finally, we conclude in Section 7.

2. Related Work

Microservices emerged in 2014 (when Lewis and Fowler first defined the microservice-based architectural style) [3]. Since then, microservices have been steadily increasing in popularity. Researchers focused on deriving Systematic Mapping Studies (SMS) on microservices in different aspects of analysis for their architecture.

Alshuqayran et al. conducted a study in [4] on 33 articles published between 2014 and 2016 of microservices architectures and their implementation. They focused on identifying architectural challenges, the architectural views, and quality attributes related to microservice systems. Although the authors paid attention to the qualitative and quantitative methods, they did not consider other architectural considerations of microservice architecture, such as architecture degradation challenges and root cause analysis.

Pahl et al. selected 21 articles published over 2014 and 2015 to construct a characterization framework to classify and compare microservice-centric articles in [5]. This study resulted in a knowledge base of current research approaches, methods, techniques, best practices, and microservice architecture experiences. Following a common ontology in [6], these framework characteristics were categorized by three central concepts (computational entity, purpose, and quality).

Both of these above studies reveal that microservices research is still in a formative stage. More experimental and empirical evaluation of the benefits is needed to fill wide gaps between the current industry level and academia. Moreover, a weak research dataset did not consider crucial attributes for architecture analysis, such as security and dynamic analysis aspects.

Since microservices come from practitioners and research comes later, Soldani et al. in [3] aimed at complementing the academic studies with a systematic analysis of the industrial gray literature on the topic. They performed a deep analysis on classifying the technical and operational pains and gains of microservices as recognized by industrial researchers and practitioners working day-to-day with microservices. This analysis included 51 selected industrial studies published from 2014 until the very end of 2017. It showed that the understanding of the pains and gains of microservices is quite mature in the industry, but academia has much to learn from the industry on the topic. The identified drawbacks are mainly related to the intrinsic complexity of microservice-based applications, while the gains relate to peculiar properties of microservice-based architectures.

Continuing to compare industrial and researching different views, Di Francesco et al. in [7] selected 71 primary studies up to 2016. They extended it in [8] to include 103 relevant papers until the beginning of May 2017. Their objective is to identify, classify, and evaluate the focus for industrial adoption of existing research in architecting with microservices from a researcher's and practitioner's point of view. This work contributes with (1) a classification framework for research studies on architecting with microservices, (2) a systematic map of current research of the field, and (3) an evaluation of the potential for industrial adoption of research results. Finally, the authors investigated the tradeoff between flexibility and complexity calls for intensive interactions; then, they showed that significant microservice-based systems must consist of much larger numbers of microservices than the small examples covered by the publications so far.

Taibi et al. [9] organized different microservice patterns in a catalog of patterns. They selected 42 papers published between 2014 and 2016. They created a three-layered catalog of patterns consisting of Data Storage Patterns, Deployment Strategies and Patterns, and Orchestration and Coordination Architecture Patterns. This catalog shows some patterns that were clearly used for migrating existing monolithic applications (service registry pattern) and others for migrating existing SOA applications (hybrid pattern), adopting the API-Gateway pattern in the orchestration layer to benefit from microservice architectures without refactoring. However, this study considered case studies suitable patterns without considering the quality attributes that may impact such decisions. Therefore, there is a lack of understanding of how to adopt a microservice architecture style in practice.

DevOps tries to support developers with a set of continuous delivery practices and tools to continuously deliver value, increasing delivery efficiency and reducing the time intervals between releases [10]. Microservices emerge as an architectural style. However, in short order, they extended into deployment and operations as a DevOps style. The above studies showed a lack of tool support to automate and facilitate cloud microservice and deployment.

Taibi et al. extended their former study [9] to put more focus on how to continuously deliver value in a DevOps pipeline in [10]. This extension includes 23 articles published in the same period between 2014 and 2016. They focused on analyzing the different microservice architectural styles to map existing tools and techniques adopted in the DevOps context. They considered the four quality attributes (availability, modifiability, performance, and testability) as well. For that reason, their study focused on a set of tools that enables continuous integration, test automation, rapid deployment, and a synchronized environment. Additionally, they identified phase-based techniques as follows: Planning and Coding Techniques, Testing Techniques, Release Techniques, Deployment Techniques, and Operation and Monitoring Techniques.

Waseem et al. in [11] continued in considering DevOps in microservices architecture. They selected 47 articles published from 2015 to 2018. This study aimed to identify problems, solutions, challenges, patterns, qualities attributes, tools, application domains, and research opportunities in the context of microservice architecture in DevOps. They classified this study into three major themes: "microservices development and operations in DevOps", "approaches and tool support in DevOps", and "microservice architecture migration experiences in DevOps". In addition, they identified 50 tools that support

building such architecture-based systems in DevOps, for instance, GitHub as a version control system, Jenkins as a continuous integration server, and Puppet as a configuration management system, which are the most popular tools.

The aggregated research questions shown in Table 1 show that the above studies have covered work on microservices relating to the challenges and techniques of developing microservices. However, there has not been a work to cover techniques relating to analyzing a microservice system as it currently exists, and while the design of microservice architecture has been covered in these studies, the reconstruction and analysis of that architecture have not. Furthermore, these studies have not yet covered the most recent years, with the latest study covering up to 2018. It is these edges that we aim to address with our mapping study.

The audience of our study consists of both researchers interested in investigating the microservices architectural style and practitioners that are willing to understand and adapt existing research on microservice architecture.

Table 1. Related Work Research Questions.

Reference	Research Questions
[3]	How much evidence of microservices experimentation from industry is available online? What are the technical and operational "pains" of microservices? What are the technical and operational "gains" of microservices?
[4]	What are the architectural challenges that microservices systems face? What architectural diagrams/views are used to represent microservices architectures? What quality attributes related to microservices are presented in the literature?
[5]	What are the main practical motivations behind using microservices? What are the different types of microservice architectures involved? What are the existing methods, techniques and tool support to enable microservice architecture development and operation? What are the existing research issues and what should be the future research agenda?
[7,8]	What are the publication trends of research studies about architecting with microservices? What is the focus of research on architecting with microservices? What is the potential for industrial adoption of existing research on architecting with microservices?
[9]	Which are the different microservices-based architecture patterns? Which advantages and disadvantages have been highlighted for these patterns?
[10]	Which are the different microservices-based architectural styles? What are the differences among the existing architectural styles? Which advantages and disadvantages have been highlighted in implementations described in the literature for the identified architectural styles What are the different DevOps-related techniques applied in the microservices context?
[12]	What is the frequency and type of published research on microservices in DevOps? What are the existing research themes on microservices in DevOps and how can they be classified and mapped? What problems have been reported when implementing microservices in DevOps? What solutions have been employed to address the problems? What challenges have been reported when implementing microservices in DevOps? What methods are used to describe microservices in DevOps? What microservices design patterns are used in DevOps? What quality attributes are affected when employing microservices in DevOps? What tools are available to support microservices in DevOps? What are the application domains that employ microservices in DevOps?

3. Methods and Materials

The motivation behind this study was to understand the current landscape of how microservices are analyzed. We aimed to identify and assess the current methods in use and recognize the specific goals that fall under the purview of microservice analysis. We also desired to know whether the analysis of microservice architecture was related to other architectures or if it was too different for existing techniques to be reused. To this end, we defined these research questions:

RQ1 What methods and techniques are used in microservice analysis?
RQ2 What are the problems or opportunities that are addressed using microservice analysis techniques?
RQ3 Does microservice analysis overlap with other areas of software analysis, or are new methods or paradigms needed?
RQ4 What potential future research directions are open in the area of microservice analysis?

To perform this study, we used the guidelines for software engineering mapping studies proposed by Petersen et al. [13]. The set of research questions above defined the bounds and goals of the study. Next, we crafted a query to search for relevant works to answer these questions, manually filtering out results we found not applicable, with the inclusion and exclusion criteria defined below. Finally, we analyzed the remaining results and extracted a categorization of the works to answer our research questions.

Next, we defined our query. The primary terms are "*microservice*" or "*microservices*". We chose secondary terms to identify those results that addressed "*analysis*" or "*analyzing*" and architectural challenges in microservices. In particular, for architectural challenges, we searched for works that address issues of "*architecture degradation*", equivalently called "*architecture erosion*" or "*architecture degeneration*" in the literature. Next, we considered works on "*architecture reconstruction*" of microservice systems. We further identified works that addressed "*technical debt*" in microservices, as this is closely linked to changes in the architecture. Thus, our query was structured as follows:

```
    ("microservice" OR "microservices")
AND
    ("analysis" OR "analyzing" OR "architecture reconstruction"
        OR "architecture degradation" OR "architecture erosion"
        OR "architecture degeneration" OR "technical debt")
```

To obtain the most recent and relevant results, and to narrow the scope of the research, we limited the search results only to the years 2018–2021. We applied this query to six major indexing sites: the ACM Digital Library, IEEE Xplore, SpringerLink, Elsevier ScienceDirect, MDPI, and Wiley.

From the initial results, we manually vetted the works further based on their actual relevance. Specifically, we applied the following *inclusion criteria* to determine if a work belonged in the study:

1. Papers that performed program analysis on microservice-based systems in some capacity with the goal of extracting some information about the system.
2. Papers with analysis that was designed for or was being discussed in the context of microservices specifically.
3. Papers that discussed and addressed challenges associated with microservice analysis.
4. Papers that described a benchmark microservice system intended for use as a testbed.
5. Papers with full text available in the selected databases.
6. Papers published in last three years (2018–2021).

We further applied the following *exclusion criteria* to eliminate works not relevant:

1. Papers that presented tools or methods designed to directly assist or participate in the operation of the microservice system (as opposed to strictly analyzing and extracting information from it).
2. Papers describing a specific implementation of a real-world microservice system not intended as a benchmark.
3. Papers without specific output, suggestions, or opinions regarding microservice analysis without experiments or robust proposed methods.
4. Papers without full text available in the selected databases.
5. Papers not presented in the English language.
6. Papers not from peer-reviewed sources.

The initial and manually filtered search results of the query are shown in Table 2.

Table 2. Search results from the query applied across scientific indexers, all results and filtered results assessed in this study. Refer to Table 3 for specific works and classification.

Indexer	All Search Results	Filtered Works Used in This Study
ACM DL	785	11
IEEE Xplore	364	17
ScienceDirect	650	6
SpringerLink *	368	19
MDPI	28	1
Wiley	13	1
Total	2208	55

* Limited to available full texts.

Table 3. Selected paper categorization.

References	Approaches and Tools					Challenges and Goals							
	Static Analysis	Dynamic Analysis	Graph-Based Analysis	Model-Based Analysis	Pattern-Based Analysis	Architectural Analysis	Microservice Migration	SAR	Technical Debt Analysis	Quality Attribute Analysis	Fault Analysis	Evolution	Surveys
Cerny et al. [14]	✓				✓								✓
Baresi [15]	✓					✓							
Saidani [16]	✓					✓							
Kamimura et al. [17]	✓					✓							
Furda et al. [18]	✓					✓							
De Alwis et al. [19]	✓	✓				✓							
De Alwis et al. [20]	✓	✓				✓							
Matias et al. [21]	✓	✓				✓							
Soldani et al. [22]	✓	✓										✓	
Eski et al. [23]	✓	✓	✓			✓							
Ren et al. [24]	✓	✓	✓			✓							
Ma et al. [25]	✓		✓				✓				✓		
Walker et al. [26]	✓		✓				✓					✓	
Walker et al. [27]	✓		✓		✓							✓	
Walker et al. [28]	✓		✓		✓							✓	
Pigazzini et al. [29]	✓				✓			✓				✓	
Marquez et al. [30]	✓				✓				✓		✓		
Tighilt et al. [31]	✓				✓							✓	
Kecskemeti et al. [32]		✓				✓							
Jin et al. [33]		✓				✓							
Kleehaus et al. [34]		✓					✓					✓	
Jiang et al. [35]		✓							✓				
Somashekar et al. [36]		✓							✓				
Brondolin et al. [12]		✓							✓				
Ravichandiran et al. [37]		✓							✓				
Chen et al. [38]		✓							✓		✓		
Bogner et al. [39]		✓							✓			✓	
Samardzic et al. [40]		✓							✓			✓	
Du et al. [41]		✓									✓		
Meng et al. [42]		✓									✓		
Jin et al. [43]		✓									✓		
Zuo et al. [44]		✓									✓		
Zhou et al. [45]		✓									✓		✓
Guo et al. [46]		✓	✓								✓		
Brandon et al. [47]			✓			✓					✓		
Christoforou et al. [48]			✓				✓						
Nunes et al. [49]			✓				✓						
Li et al. [50]			✓				✓						
Stojanovic et al. [51]			✓				✓						
Guan et al. [52]			✓								✓		
Liu et al. [53]			✓								✓		
Meng et al. [54]			✓								✓		
Wu et al. [55]			✓								✓		
Ma et al. [2]			✓								✓	✓	
Sun et al. [56]				✓		✓							
Rademacher [57]				✓			✓					✓	
Khazaei et al. [58]				✓					✓				
Chondamrongkul et al. [59]				✓					✓				
Klinaku et al. [60]				✓					✓				
Mendonca et al. [61]				✓	✓								
Mczara et al. [62]				✓	✓								
Toledo et al. [63]					✓			✓				✓	✓
Cerny et al. [1]					✓								✓
Auer et al. [64]						✓							✓
Bogner et al. [65]								✓				✓	✓

4. Review of the Selected Studies

After filtering our results down to papers relevant to microservices and the scope of this study, we read and organized them according to the goals and methods that were most common among them. We considered the papers based on two different perspectives: *approaches and tools* and *challenges and goals*. The specific perspective categories are not exclusive, as many papers use a combined approach to address some goals. The categories we identified are summarized in Table 3 also showing which papers fit in which category. In the remainder of this section, we describe the particular categories, grouped by the perspectives.

4.1. Approaches and Tools

The papers we assessed used a variety of methods and techniques to reach their goals. In particular, we have identified common approaches to static and dynamic analysis. Static analysis considers examining the application source code without the need to run the system. On the other hand, dynamic analysis requires the system to run to extract information. In addition, some papers also combined static and dynamic analysis for their goals. Another identified approach bases on model-based analysis. In this approach, models are constructed as the system focal point to understand the system or to use the model by automated tools to derive details about the system. Furthermore, what is also common across researchers is to use graph-based analysis to represent the microservice-based system architecture in the form of a graph and then operate with the graph rather than with a complex system. Finally, we found another rather common approach based on patterns. However, it is always combined with another approach. Both the model-based and graph-based analysis introduce an abstraction or an intermediate system representation to capture certain system concerns. Although there might be an overlap or a graph can be understood as a model, we consider both categories separate.

4.1.1. Static Analysis

Static analysis involves examining the source code or bytecode of the application without deploying them. What was most interesting to note about papers that used static-code analysis was their frequent use to gather information about microservice architecture. In an article from Marquez et al. [30], the authors analyzed source code from 17 different microservice systems on GitHub by reviewing each system's documentation and examining the structure of the source code to find any common patterns. From this analysis, the authors obtained a set of availability tactics, also known as resiliency patterns. These patterns are meant to provide predefined designs to best counter common problems in the architecture. An article by Tighilt et al. [31] performs a similar analysis with 67 microservice systems and obtains a set of antipatterns from their results. In this situation, their use of static-code analysis also allowed them to identify concrete refactoring solutions that would counter these antipatterns. A paper by Pigazzini et al. [29] also uses static-code analysis for improving microservice architecture by implementing detection methods for three different microservice smells in a pre-existing tool called Arcan. Baresi et al. [15] iteratively map OpenAPI specifications to concepts in reference vocabulary, which are computed to be the best matches. Then each candidate microservice, one per reference concept, is defined by its operations and their parameters, complex types, and return values. Saidani et al. [16] use static analysis to identify candidate microservices while considering structural dependencies of the source code. First, an empty set of microservices is created. Next, each class is assigned a microservice. Lastly, a genetic algorithm called NSGA-II is used to determine the best tradeoff between cohesion and coupling. The result is a tool called MSExtractor that is shown to outperform recent extraction approaches. A paper by Walker et al. [26] uses static-code analysis, both source code and bytecode, as well as other dependency and application analysis to construct bounded contexts of the different views—or sets of related artifacts that cover a concern of the architecture of the system—for each microservice. Then it aggregates them into a full-scope centralized perspective for each view, which

consists of all the microservices aggregated into a mesh. The same principle was later extended to detect code smells searching for patterns [27,28]. Kamimura et al. used source code [17] to extract microservice candidates from monolith applications. The authors used the SarF clustering method [66] to eliminate specialists' manual effort in analyzing different architectural views. Furda et al. [18] used static analysis to identify possible data leaks among microservices while migrating from monolithic applications.

4.1.2. Dynamic Analysis

Dynamic or runtime analysis is an alternative to static-code analysis. For instance, two papers use a black-box dynamic analysis to gather results. This meant they focused on the measurable results of the application rather than the source code, with a framework proposed by Brondolin et al. [12] collecting data based on low-level performance (e.g., cycles, Instruction Retired (IR) events, cache references, cache misses, and power consumption), application performance (e.g., CPU usage, execution time) and network activity (e.g., number of requests, bytes sent and received, and average latency) to better monitor the transparency, performance, and accuracy of cloud-based microservice applications. Somashekar et al. [36] also take this black-box approach to evaluate various optimization algorithms for microservice systems, which are meant to minimize the tail latency of the given microservice application deployment. By taking this approach, the authors find out which optimization algorithms are the most efficient for reducing a system's latency. An article by Bogner et al. [39] proposes a dynamic approach to calculate a set of 58 service-based maintainability metrics proposed in scientific literature using data gathered at runtime. A software architecture recovery framework developed by Kleehaus et al. [34] known as MICROLYZE works towards its goal mainly by gathering infrastructure information about a microservice-based system using both a service discovery tool and a distributed tracing solution, then monitoring incoming requests on the system and mapping them to business processes. Kecskemeti et al. [32] incorporate several techniques in the process of microservice creation. First, a recipe-based generic image creation service is presented that can create VM and container images crafted for particular cloud systems. Next, a dynamic, live-evaluation-based image size optimization technique is used to create a family of images based on the previous monolithic service. Lastly, this image family is turned into a set of microservices within the ENTICE environment. Du et al. [41] use dynamic analysis and machine learning to build an anomaly detection system consisting of three modules, monitoring, data processing, and fault injection. The monitoring module collects performance data from the target system, which is processed by the data processing module to detect anomalies. Lastly, the fault injection module simulates service faults and gathers datasets of performance data representing the normal and abnormal conditions which are then used to train the machine learning models. Ravichandran et al. [37] also used the system's resource metric to evaluate resource behavior in autoscaling systems of microservices to discover security anomalies. The authors put the mechanism to the test on the SAVI testbed and observed that it could successfully detect anomalous application behavior [67]. Chen et al. [38] leveraged resource metrics to find anomalies in the streaming of microservice systems.

There are other forms of dynamic analysis commonly seen in papers on microservices. Two common specializations are log analysis and analysis of execution traces. A proposed framework by Zuo et al. [44] uses both techniques to detect anomalies in a system, referring to logs as temporal data and query traces as spatial data. With this framework, a system's execution logs are first processed with an online template extraction method to obtain service execution behaviors. They use the information gathered from logs alongside queries being traced from a user's frontend call all the way to queries made on a system backend, which is handled with tools such as Dapper, Zipkin, and Osoprofiler. The information collected from both log analysis and query traces is then used to unveil a system's hidden systematic status, which is then used in an anomaly detector based on a one-class classifier. Jiang et al.'s proposed scheme [35] uses multiple trackers on logs, sensors, and internal

metrics to set up an intelligent microservice monitoring scheme, which can help keep track of anomalies in a system and warn clients of these anomalies before they develop into failure cases. An article by Samardzic et al. [40] uses log analysis as well, basing its results on runtime logs collected from six real-world microservices commonly used in the retail domain to determine how microservice runtime performance degrades. An algorithm proposed by Jin et al. [43] relies on execution traces. It first screens a potential anomalous time period during a system's runtime. Then, it generates a method invocation chain of the system. The proposed framework then traces back through the anomalous nodes in the invocation chain to determine the most likely root causes. Jin et al. [33] proposed a method based on functionality-oriented microservice extraction (FoME) in their study, which monitors the program's dynamic behavior and clusters execution traces to group source code entities. A tool built by Meng et al. [42] uses mining system call patterns to diagnose faults in a microservice system. First, the system calls are collected into sequences, which are then clustered into sequence patterns. Then a GRU-based neural network is adopted to model a sequence pattern to predict the future system call. Faults are then identified by comparing the predicted system call to the actual one.

4.1.3. Combined Dynamic and Static Analysis

Some papers use a combination of static and dynamic analysis to achieve their results. A technique used by De Alwis et al. [19] to discover potential microservices within enterprise system components first statically analyzed the system to determine the business objects it manipulates, then executes the system to generate and extract data related to system execution, such as logs and traces. A paper by Matias et al. [21] uses this hybrid static-dynamic approach to determine boundaries of microservices decomposed from a monolithic system. First, they statically analyze a system and use the collected information to generate a graph of the system. Then, by monitoring the system at runtime to gather operational data, they identify how the dependencies are exercised during execution, and gain an understanding of how the system is actually used. Another article by De Alwis et al. [20] uses the static-dynamic hybrid to modularize monolithic enterprise systems for applications in Industrial Internet of Things networks, then extracts crucial dependencies from these analysis methods. Eski et al. [23] analyzed static codes using both static and dynamic analysis to construct a software system as a graph that shows different relations and couplings between the classes to assist the migration of monolithic to microservice architecture. Ren et al. [24] also address the migration of legacy monolithic applications and their data to microservices architecture. They employed static and dynamic analysis to achieve the static structure in addition to runtime behavior characteristics of a monolithic application. Soldani et al. [22] combined static and dynamic mining of information of a microservice system to obtain a methodology that identifies architectural smells. The authors used Kubernetes deployment files instead of source code for static mining.

4.1.4. Model-Based Analysis

Another common analysis technique we found was model-based analysis. These models were meant to be understood more by automated systems and researchers than by developers. In an article made by Rademacher et al. [57] a modeling language is created that is meant to systematize Software Architectural Reconstruction (SAR) of microservice-based systems. Another article by McZara et al. [62] models the dependencies between microservices using a design structure matrix, commonly used as a simple representation of a complex application. This matrix can then be analyzed to determine which microservice dependencies should be prioritized to build a system with higher resilience against errors. Chondamrongkul et al.'s proposed framework [59] looks at a design model of a microservice system that an automatic security analysis system can analyze to determine weak points and show how attacks may occur on the system. An article by Mendonca et al. [61] shows Continuous-Time Markov Chain (CTMC) models of various developer resiliency patterns, which are used to make a system more resilient against errors and failure states.

Then analyzes these models using a probabilistic model checker known as PRISM. In a study by Khazaei et al. [58], the authors first creates a microservice platform on Amazon EC2 cloud using Docker, then use the results from experiments run on this platform to establish a tractable analytical performance model that can be used to perform what-if analysis and capacity planning in a systematic manner for large scale microservices with a minimum amount of time and cost. Stojanovic et al. [51] proposed a method to assess migration of monolith application into microservices where it analyzes data-flow diagram and identifies primitive functions that communicate with each other using data stores. Sun et al. [56] use both model-based analysis and static analysis in their paper about decomposing monolithic Internet of Things systems into microservices, with both a top-down approach via analysis of the system's domain model and a bottom-up approach via static analysis of source code. Klinaku et al. in [60] uses model-driven architectural simulator called Palladio [68]. It helps in analyzing and predicting quality attributes of system architectures.

4.1.5. Graph-Based Analysis

Another common technique we found was graph-based analysis, representing the architecture of a microservice-based system in the form of a graph. Graph-based analysis can fall into either static or dynamic analysis categories based on the source of graph creation. For example, if the graph is created from the source code, it is a static analysis. In contrast, creating the graph from logs or network connections is dynamic analysis. A representation of microservices based on a graph has an advantage in easy visualization and being able to naturally represent the graph-like connections between the services. Graph-based analysis was used most commonly in papers to handle root cause analysis, with papers by Brandon et al. [47], Wu et al. and Meng et al. [54,55] creating graphs based on a microservice system in an erroneous state, using nodes representing the clients, hosts, databases, etc. and then tracing back through the created graph to determine an error's most likely root causes. An article by Guan et al. [52] also uses graph-based analysis for handling root cause analysis by first generating a causal graph of the system using an application called Microscope. Their proposed algorithm for Root Cause Analysis then traverses over the created graph. An article by [46] first describes gathering span logs from an execution trace of a system, then using this data to assemble a graph with traces, paths, and business flows. This graph is meant to make it easier for developers to understand the architecture of their microservice systems and diagnose future problems. Ma et al. [2] create a similar graph of a microservice system meant to visualize microservices. An article by Li et al. [50] on the other hand, primarily breaks down a monolithic architecture into a data-flow diagram (a specific kind of graph) and uses the diagram to determine the best candidates for a microservice translation. A tracing and analytics tool called JCallGraph developed by Liu et al. [53] uses graph-based analysis as a tool for developers who need to monitor and debug their systems, which works by tracing the invocations of microservices by other microservices in a system and assembling an invocation graph representation of the system. Walker et al. [27,28] achieve their goal of code smell detection by performing graph-based static analysis. First, a graph is generated that shows the different interactions between the microservices by exploring each microservice for a connection to other microservices. The dependency management tools for each microservice and the application configuration are then analyzed and processed to detect if any of 11 selected microservice smells are detected. Similarly, Walker et al. [26] use this same approach to extract system architecture. Christoforou et al. [48] achieve their goal of assisting with migration to microservices by identifying key concepts and organizing them as a Multi-Layer Fuzzy Cognitive Map. Static analysis is performed on the model to allow the identification of strong concepts by determining the complexity of the graph, the weight and number of edges of each node, and the tendency of cycles. Next, dynamic analysis is performed on the model during execution through simulations of manually configured scenarios. This allows the concepts identified by the static analysis to be ranked based on their significance. Nunes et al. [49]

use an approach based on static analysis of monolithic source code to develop a text-call graph. Next, the call graph is used to generate a dendrogram, which is then cut into a set of clusters. These clusters are then used for visualization and modeling to determine the best decomposition. Eski et al. [23] analyzed the way to migrate systems from monolithic to microservices architecture. They applied graph clustering techniques to the analysis results to identify microservice candidates. Ren et al. [24] approached the migration of legacy monolithic applications to a microservices architecture-based one. They represented the analysis results into graphs with the degree of dependence between functions. After that, they cluster these functions to achieve the required migration. Ma et al. [25] proposed the technique GMAT (Graph-based Microservice Analysis and Testing), which creates a service dependency graph to visualize microservices and analyze error root detection using a combination of static and graph-based analysis.

4.1.6. Pattern-Based Analysis

Many of the articles we processed considered identifying development patterns [27,29–31,61,62]. Pattern-based analysis mostly employs static analysis to recognize specific patterns in the source code [27,29,31]. Moreover, architectural patterns and mathematical model checking approaches have also been explored in related studies [30,61,62]. Pigazzini et al. [29] demonstrated the detection of three code smells in microservices: Cyclic Dependency, Hardcoded Endpoints, and Shared Persistence. Their approach involves static analysis of source code to match predefined patterns to recognize those smells. The authors validated their works on five open-source projects and further discussed the roadmap for implementing the next set of smells. A similar study was conducted by Walker et al. [27,28] where the authors identified 11 microservice-specific code smells using static analysis. They also used several adjustable threshold values to change the flexibility of pattern matching. They presented an open-source tool MSANose to demonstrate their approach. Tight et al. [31] proposed a catalog of 16 microservice antipatterns. The authors grouped these antipatterns into four categories: design, implementation, deployment, and monitoring. Their work is based on an analysis of 67 open-source projects and 27 related studies. Márquez et al. [30] focused on availability tactics that provide architectural solutions to address several security concerns. They examined the source code and documentation of 17 open-source applications and identified 5 uses of availability tactics. Moreover, they introduced a template or pattern for characterizing these availability tactics. Mendonca et al. [61] examined resiliency patterns in service-to-service interactions, more specifically, retry and circuit breaker patterns. They presented a probabilistic model checker to investigate the course of resiliency patterns as continuous-time Markov chains (CTMC). McZara et al. [62] modeled an outage-impacted microservice system, concentrating on the DevSecOps domain. They used an existing dependency analysis pattern called a Design Structure Matrix (DSM) to identify critical links among the microservices.

4.1.7. Tools

This section summarizes the tools we have identified across assessed papers. These studies either developed new tools or used existing tools for their analysis. Table 4 enumerates the tools that we found across the papers. Arcan is an existing static analyzer tool that has been used by Pigazzini et al. [29] to detect code smells. For a similar purpose, Walker et al. developed a new tool MSANose [27] that can identify 11 microservice-specific code smells. The tool MSExtractor was presented by Saidani et al. [16] to decompose monolithic applications into microservices. Kleehaus et al. introduced the tool MICROLYZE, which can recover software architecture by mapping infrastructure information with business processes. Dapper, Zipkin, and Osoprofiler are exiting query tracing tools that were used by Zuo et al. [44] to trace calls from the application's frontend to the backend. Microscope is an existing graph generation tool that was used by Guan et al. [52] for root cause analysis. JCallGraph is a newly developed tool by Liu et al. [53] for tracing and monitoring using the graph-based approach. μTOSCA is proposed and developed by Saldana et al. [22]

to analyze and detect architectural smells. Klinaku et al. [60] used the existing tool Palladio [68] to conduct a case study to analyze the performance of microservice applications.

Table 4. Identified tools.

Tool	Reference	Newly Developed	Purpose
Arcan	[29]	No	Code smells detection
MSExtractor	[16]	Yes	Monolithic decomposition
MSANose	[27]	Yes	Code smells detection
MICROLYZE	[34]	Yes	Architecture reconstruction
Dapper	[44]	No	Query tracing
Zipkin	[44]	No	Query tracing
Osoprofiler	[44]	No	Query tracing
Microscope	[52]	No	Graph generation
JCallGraph	[53]	Yes	Tracing and analytics
μTOSCA	[22]	Yes	Architectural smells detection
Palladio	[68]	No	Performance analysis

4.2. Challenges and Goals

We found that current research, regardless of the approach used, often addresses multiple overarching categories. Across the assessed papers, various challenges and goals were approached. We first identified the goal of architectural analysis, taking into account pre-existing microservices system and intends to improve its qualities or detect an issue. Software architecture reconstruction is a related goal intending to explain the existing system. Migration from monolithic systems to microservices is also common. Another challenge gaining popularity is technical debt analysis, attempting to identify inadequate solutions applied in the system, which will lead to system quality deterioration. We identified works addressing challenging system qualities, including security or performance. Other works targeted fault analysis and often specifically root cause analysis. Finally, we identified surveys intending to identify state-of-the-art practices related to microservice analysis. These challenges are described in the following text.

4.2.1. Architectural Analysis

Most of these papers use architectural analysis to address a specific goal within pre-existing microservice architectures, such as improving security or detecting faults. This is often done by creating tools or frameworks that take a white-box approach to analyze the architecture. Some papers, however, do not focus on analyzing the architectures of microservices. The scopes of these papers tend to be broader, and most of them do not present a new tool or framework. However, of those that do, the approach is often black-box and serves goals not specific to a single microservice, such as identifying the optimal configuration for a microservice application. A few papers discussed key differences between analyzing microservice architecture in comparison to other architectures. Cerny et al. [14] discuss some pitfalls that traditional code analysis has with addressing microservice systems, namely that it overlooks important enterprise development framework constructs, which are key for building enterprise microservice systems. Some papers treat the relationships between Service-Oriented Architecture (SOA) and microservices differently. For instance, Brandon et al. [47] does not necessarily distinguish differences between SOA and Microservices since the paper's proposed approach of graph-based root cause analysis can be used on both types of systems. Cerny et al. [1] discusses the key differences between SOA, microservices, and Self-Contained Systems (SCS). One of the major differences it mentioned was that services are brought to production independently of each other in a microservice architecture, whereas this is not the case with most SOA solutions. Toledo et al. [63] begins by comparing SOA and microservice architectures. For example, there are concepts and techniques from microservice architecture that were borrowed from SOA, such as scalability. Microservices, however, differ in their emphasis

on service granularity and the fact that they have two types, functional or infrastructure, while SOA can have several types. Serverless architecture has also been studied in the context of microservices.

4.2.2. Migration to Microservices

Some works aim to provide tools to assist in the process of migrating from monolithic or service-based architectures to microservice-based ones. Baresi et al. [15] aim to aid in identifying potential microservice candidates by using OpenAPI specifications and reference vocabulary to identify potential candidate microservices. Saidani et al. [16] introduce MSExtractor, a novel approach that uses a non-dominated sorting genetic algorithm (NSGA-II) to decompose an object-oriented application into cohesive, loosely coupled microservices. This tool uses static-code analysis to determine a decomposition that is optimized with respect to two objective functions: cohesion and coupling [24]. A systematic methodology proposed by Li et al. [50] identifies clusters accessing the same datastore and merges them with duplicate processes to create candidates for new microservices. Kecskemeti et al. [32] propose a methodology based on dynamic analysis to divide a monolithic service into smaller microservices, increasing the elasticity of large applications and allowing more flexible composition with other services. Sun et al. [56] aim to create a better framework, based on model analysis that decomposes a monolithic IoT system into a microservice-based IoT system. A case study performed on an unmanned aerial vehicle (UAV) system shows that the UAV can be dynamically reconfigured to handle runtime changes. Research done by Dealwis et al. [19] develops a technique that uses static and dynamic analysis to support the re-engineering of an enterprise system based on the fundamental mechanisms for structuring its architecture. Now, two years later, Deals et al. [20] present a software modularization technique for enterprise systems to support the discovery of fine-grained microservices, which can be extracted and embedded to run on Industrial Internet of Things network nodes. Eski et al. in [23] reached a success rate of their migration approach of 89% when they measured similarity between candidates and actual services. They recommend an improvement of assigning weights to graph edges to increase the accuracy of extraction and determining thresholds for sub-clustering the constructed graph. Ren et al. [24] employed static and dynamic analysis to cluster the system functions for achieving the required migration. Finally, they performed experiments to verify the performance and scalability of their approach, and the experimental results show that the proposed method can efficiently migrate legacy applications.

Creating new tools and methods to assist in deciding if migration to a microservice-based architecture would be beneficial is also a common research direction. Christoforou et al. [48] propose the first Decision Support System to support migration to microservice-based systems using a graph-based method. The resulting tool assists in determining if the system meets the conditions to migrate and will actually benefit from it. Similarly, the paper by Auer et al. [64] surveys developers to create a framework to assist companies in determining if they should migrate their monolithic system to a microservice-based one. Cerny et al. [14] explain the potential pitfalls of converting a system from a monolithic architecture to a microservice architecture while also discussing the value of static-code analysis in microservice systems. Nunes et al. [49] aim to assist the decomposition of monolithic applications to microservice-based ones by proposing the decomposition be done by identifying where a business transaction is divided into several transactional contexts. This identification of transactional contexts groups domain entities by where they are executed instead of by their structural domain inter-relationships, allowing developers to determine the impacts of decomposition. The research goal of Matias et al. [21] is to determine the best ways to set boundaries on microservices decomposed from a monolithic enterprise system through static and dynamic analysis of the system. Furthermore, Stojanovic et al. proposed a structured system analysis to assist system migration to microservices [51]. Jin et al. demonstrated a method of migration from monolith to microservices using the functionality of execution trace [33], whereas Kamimura et al. [17] proposed this migra-

tion using source code analysis. Furda et al. [18] described an approach to identify data interference defects while migrating from monolithic to microservices. They explain the data interference problem based on information flow control theory that can cause data leaks among the microservice components. Furthermore, they developed a tool for PHP applications to demonstrate their proposed solution using static-code analysis.

4.2.3. Software Architecture Reconstruction (SAR)

In some studies, the main goal is microservice architectural reconstruction, which involves monitoring service interaction as the architecture grows and evolves over time. A framework proposed by Kleehaus et al. [34] uses dynamic analysis to facilitate the reconstruction of an microservice system's architecture with service discovery tools and to keep track of how new microservices emerge over time. Rademacher et al. [57] systematize the reconstruction of an MSA-based system's architecture by first gathering architecture information from architecture-related artifacts or views, then transforming these views into a canonical representation of a system by storing it into a database. Walker et al. [26] propose a method for automatically completing Software Architectural Reconstruction (SAR) of a microservice system through code analysis and demonstrating it on a case study on an existing microservice benchmark application. Furthermore, the work of Ma et al. can visualize architecture in addition to performing root cause analysis using static-code analysis [25].

4.2.4. Technical Debt Analysis

Another goal commonly found in papers is identifying and resolving cases of technical debt, the extra work that results from choosing a sub-optimal solution. Toledo et al. [63] propose techniques to assist in detecting and solving cases of architectural, technical debt across different stages of development in microservice-based applications through a survey of employees at companies who work with microservices. The paper by Pigazzini et al. [29] defines a detection strategy for known code smells that degrade the development of a system, such as shared persistence, hardcoded endpoints, and cyclic dependency. This strategy is based on static-code analysis and makes use of a dependency graph generated by Arcan.

4.2.5. Microservice Evolution

Microservice architecture is notable for being one that facilitates a rapid pace of development, leading to a quickly evolving architecture. Thus, tools and techniques are needed to manage this rapid evolution, both to verify that the current state of the architecture is acceptable and to ensure that the system remains maintainable and sustainable throughout the evolutionary process.

To this end, several techniques exist in the literature for addressing architecture evolution. As mentioned above, several works aim to perform Software Architectural Reconstruction on microservice systems, and these techniques also apply to evolution. Kleehaus et al. [34] specifically seek to address issues of continuing documentation of new and evolving microservices that emerge during development. The approach proposed by Rademacher et al. [57] similarly addresses the evolution of the microservice system using SAR and further develops it by integrating SAR with their previously developed microservice-specific modeling language, which allows for easier documentation and representation of microservice architecture. The SAR performed by Walker et al. [26] follows four specific architectural views, each of which provides a different viewpoint of how a microservice system evolves: the service viewpoint shows how the services and their relationships evolve, the domain viewpoint shows how the domain model used by microservices evolves, the technology viewpoint demonstrates changes in the technology stack used in the system, and the operational viewpoint shows users how their system evolves in its deployment. Relatedly, Ma et al. [2] construct a service dependency graph among microservices to visualize the evolution of architectural change, and they further

protect the evolution process by providing a mechanism to retrieve existing microservices that suit user needs to prevent unnecessary development as well as suggest test cases for regression testing to prevent damaging changes.

One threat to the sustainable evolution of microservices is poor coding design. Several works seek to address this aspect of evolution. For example, microservice antipatterns threaten to invalidate the benefits gained using microservice in the first place, and Tighilt et al. [31] analyze existing literature to identify a catalog of 16 microservice antipatterns, the definitions of which can be used to combat poor microservice design. Walker et al. [27,28], and Pigazzini et al. [29] take this further and propose methods of automatically detecting selected microservice antipatterns or code smells using static-code analysis. Works identifying technical debt are also closely related to the concept of antipatterns, and the surveys performed by Bogner et al. [65] and Toledo et al. [63] and the solutions they proposed show promise for preventing technical debt from crippling microservice evolution. Soldani et al. developed a tool, μTOSCA, to identify and resolve architectural smells [22]. μTOSCA uses the Kubernetes deployment file to resolve architectural smells using the developed tool μFRESHENER via refactoring.

Beyond issues relating to architecture and antipatterns, a number of metrics exist that indicate the health of a microservice system. These metrics provide a key insight into how maintainable a microservice system is. Bogner et al. [39] analyze runtime data of microservices to calculate several maintainability metrics relating to the microservice coupling, cohesion, complexity, and size. Samardzic et al. [40] take a different approach; they analyze performance metrics to identify performance degradation in microservices, a key indication that the current design is not sustainable.

4.2.6. Quality Attribute Analysis

Addressing system quality concerns is another common goal. Among the most common quality aspects, we identified security and performance. However, maintainability was also addressed [39].

Marquez et al. [30] use static-code analysis to systematically identify and characterize architectural tactics with regards to security in existing microservices-based systems. The tactics identified include preventing single dependency or setting timeouts and addressing security concerns such as code reuse, denial of service, or traffic between microservices. Chondamrongkul et al. [59] use model-based analysis of a system to automatically identify security threats according to a collection of formally defined security characteristics such as denial of service or man in the middle attacks. This provides an insightful result that is used to demonstrate how the attack scenarios may happen using linear temporal logic. Klinaku et al. in [60] conducted a case study using Palladio [68] to analyze the performance of microservice applications. They assessed the scalability, elasticity, and cost-efficiency aspects of a cloud-based microservice application. The authors highlighted that Palladio predicts the application performance with sufficient accuracy. However, several workarounds were needed and not automated for all the chosen scenarios when assessing these aspects.

As the number of microservices increases, the system becomes complex, and thus, it is essential to analyze the performance to identify bottlenecks in the system. Most of the studies involving performance investigation employ the dynamic analysis approach. Brandon et al. [12] described a black-box monitoring approach to analyze the performance of microservice-based applications. Their monitoring approach focused on architectural metrics, power consumption, application performance, and network performance of cloud-native applications that are deployed using Kubernetes. The authors used kernel-level instrumentation and user-space monitoring agent to collect metrics from Kubernetes workloads, and they developed a user interface to visualize the metrics. Somashekar et al. [36] investigate optimization techniques and dimensionality reduction strategies for tuning microservices applications. They demonstrated 23% latency performance improvement by configuration tuning. The authors developed an automated optimization framework that

can adopt any optimization algorithm such as genetic algorithms, Bayesian optimization, etc. Samardžić et al. [40] analyzed run time logs of six microservices in the retail area to explore performance degradation in microservices. They examined log entries to detect events and dependencies among the time series data. Jiang et al. [35] discussed how to combine the push-pull mode with SpringBoot-based microservices and then constructed a high-performance intelligent monitoring system. Khazaei et al. [58] described a performance analysis model to recognize critical components among Docker containers deployed in the Amazon EC2 cloud. From their experimental results, the authors provided a what-if analysis to minimize deployment costs and time.

Bogner et al. [39] considered dynamic analysis for maintainability. They suggested that static analysis does not fit decentralized systems. In their work, they used 23 maintainability metrics to reason about maintainability. However, their experiments need a broader assessment. Ravichandiran et al. also used dynamic resource analysis to find anomalies in security in autoscaling systems [37].

4.2.7. Fault Analysis

Analyzing faults seems to be the most common goal and direction of existing research. As microservices have become more common in recent years, finding new ways to detect and prevent faults in them is becoming increasingly important. A novel scheme based on static-code analysis introduced by Ma et al. [2] uses a graph-based method to visualize a microservice-based system. The scheme proposed generates service dependency graphs that enable developers to analyze dependency relationships between microservices and between services and scenarios. Guo et al. [46] use dynamic analysis in the form of a graph-based method to trace through the executions of requests and better diagnose various problems in microservice architectures. Various architectural tactics identified by Marquez et al. [30] solve security-related faults or prevent them from occurring in the first place. A benchmark for microservice architecture analysis proposed by Zhou et al. [45] uses execution traces to improve current industrial practices of debugging by replicating fault cases. Zuo et al. [44] provide a new general anomaly detection service based on dynamic analysis to assist in system management, using abnormal log statements to detect suspicious events and preventing failure states. An anomaly detection system provided by Du et al. [41] detects and diagnoses anomalies by addressing the problems of determining which metrics should be monitored and evaluating whether the behaviors of the application are actually anomalous or not. Meng et al. [42] propose Midiag, a sequential trace-based fault diagnosis framework that mines the patterns of microservices' system call sequences. Midiag generates predicted system calls gathered from sequence patterns based on previous system calls and compares them to the actual ones to identify possible faults. Liu et al. [53] provide a graph-based form of visual aid to developers who need to monitor and debug microservice systems, which often have tens of thousands of services. Ma et al. [25] developed a graph-based tool called GMAT that assists developers in identifying errors.

One specific concern when handling faults is performing root cause analysis when the issues arise. The paper by Brandon et al. [47] proposes a method to make it easier for developers to manage interactions between services as well as assist in detecting anomalies such as stressed hosts or stressed endpoints. Based on dynamic analysis, the method proposed is shown to be 19.41% more effective than a machine learning method at classifying the state of an anomaly to its root cause. A new experimental framework presented by Wu et al. [55] aims to better localize root causes of failures in microservice systems and detects performance issues by correlating their symptoms with system resource use. Jin et al. [43] propose an algorithm to detect time-consuming anomalies in a microservice architecture that is based on dynamic analysis of execution traces with a root cause approach in mind. Meng et al. [54] establish a new framework to find failure cases in microservice systems tested with real-world failure tickets. Guan et al. [52] design and implement a prototype to perform graph-based analysis on a microservice system in an anomalous state and determine the top candidates for the root cause of the anomaly.

4.2.8. Surveys

Several works seek to describe the state of the art and state of practice regarding microservices and their analysis. Both Toledo et al. [63] and Auer et al. [64] have conducted surveys for this purpose, with the former survey conducted among employees working with microservices to determine the most critical architectural technical debts, and the latter conducted among developers to create an assessment framework on whether or not a system should make the switch from monolithic to microservice-based architecture. Cerny et al. [14] conducted a survey as well, focused on determining whether traditional static-code analysis is fit for use in the microservice architecture. Zhou et al. [45] first use an empirical survey to gather information on challenges faced by developers and then evaluates the effectiveness of execution tracing for the purpose of debugging, using a benchmark the authors developed. Finally, a mapping study by Cerny et al. [1] is meant to gather and analyze existing research on microservices. Bogner et al. [65] created a survey based on questionnaire responses from 60 software professionals. They focused on processes, tools, and metrics used in the industry and the maintainability-related treatment of systems based on service orientation. They concluded that standard and systematic techniques have benefits for maintainability and leakage due to the absence of both architecture-level development mechanisms and the quality assurance of service-oriented approaches. The results proposed that industrial quality must be improved to avoid problems in long-living service-based software systems.

5. Results and Discussion

After assessing the identified literature and categorizing both the approaches and goals, we can answer the questions raised at the beginning of our study. The below subsections summarize and interpret the knowledge from the previous section.

5.1. RQ1: Methods and Techniques Used

There were several interesting things to note about the common trends among the types of methods. Of the two main methods of analysis, dynamic analysis is used more often than static analysis. Using dynamic techniques, several different approaches become feasible that cannot be done via static techniques, for example, performance analysis and optimization techniques as well as other metrics-based analyses [12,32,36,39,40]. A specific subset of dynamic techniques is commonly applied to fault analysis and root cause analysis: log analysis and execution trace analysis are perfect for this task, as they examine traces directly related to program execution [35,40,42–44].

Most papers that used static analysis as their main method tended to do so to gather information specifically about the architecture. These techniques depend on analyzing statically defined artifacts to reconstruct an architectural view of a system, mainly analyzing source code [16,26,28,30], but also other artifacts, such as OpenAPI specifications [15]. Static analysis is applicable to other goals as well, such as anti-pattern or code smell detection [26,28,29].

An approach used less often is that of model-based analysis, in which a specific model is built to represent the microservice system. This can range from modeling dependencies and architecture of microservices [56,57,62] to developing security models [59] or performance [60] and resilience models [58,61].

Like model-based techniques, graph-based analysis depends on representing the microservice system as a graph and then analyzing that structure, exploiting microservices' natural graph-like connections. Graph-based techniques are commonly used for detecting faults or performing root cause analysis [47,52,54,55], as well as performing monolith-to-microservice migration by representing an existing monolith as a graph that can be segmented into microservices [23,24,48,49]. However, it can also be used in tracing patterns [27,28]. Finally, graph-based methods are also used in monitoring and visualization systems [2,26,53].

Another approach is pattern-based analysis that mostly employs static analysis to recognize specific patterns in the source code [27–29,31] and architectural patterns [30]. However, it can use graphs as well [27,28]. This approach also involves mathematical model checking techniques to evaluate microservice systems [30,61,62].

The summary of our findings relating to RQ1 is shown in Table 5.

Table 5. Works relevant to RQ1.

Method	References	Total
Static analysis	[14–26,28–31]	17
Dynamic analysis	[12,19–24,32–46]	22
↪ *Log analysis*	[35,40,44,46]	↪ 4
↪ *Execution trace analysis*	[33,35,43,44]	↪ 4
Graph-based analysis	[2,25–28,46–50,52–55]	14
Model-based techniques	[51,56–62]	8
Pattern-based analysis	[27–31,61,62]	7

5.2. RQ2: Goals Addressed by the Papers

The main goal we found was that of fault analysis, either by detecting and preventing faults [2,30,41,42,44–46,53], or determining their origin through root cause analysis [43,47,52,54,55]. It is apparent that identification of the cause is important, especially given that microservices run large enterprise systems. Failures in such systems has major economic impacts, whether the users cannot access it or the cloud resource demands peak due to an error. Still, it is a challenge to perform detection in real time, and many challenges remain, as we discuss later when answering RQ4.

Another common topic addressed was migration to microservice-based architectures either by providing tools to decompose monolithic architectures into microservices [15,16,19,32,50,56] or to assist in deciding if migration is beneficial and feasible [21,48,49,64]. Although we could assume a silver bullet by now when it comes to the design of microservice systems, there is no perfect guidance for engineers managing legacy systems.

Identifying or resolving technical debt in a microservice architecture [29,39,63] is an interesting research direction of significant impact. We expect this specific research to grow significantly in the next few years as consequences of system evolution impact the operational budget.

Other goals involve analysis of the software architecture [14,28]. Several works discuss the inherent challenges in analyzing microservices versus other architectures such as monolithic systems, or SOA [1,63]. Several works propose methods of software architecture reconstruction on microservice systems [26,34,57], which is especially important to help reason about the system. Software architecture can be represented through various views serving that represent the system slightly differently, meeting the needs of distinct stakeholders.

Overlapping with technical debt and architecture analysis is the goal of streamlining and protecting the process of microservice evolution. Works performing SAR [26,34,57] combat evolutionary problems by providing the user with an up-to-date view of system architecture, giving insights that can prevent architectural degradation. Antipatterns and code smells that threaten sustainability have also been addressed [27–29,31], as has the issue of accumulating technical debt [63,65]. In addition, some works improve the evolutionary outlook of a system by addressing maintainability metrics [39] or performance degradation [40].

It is quite common to see research addressing various quality concerns of microservices such as security [30,59] or performance [12,35,36,39,40,58]. We must recall that software

architecture is the frame for various software qualities. The primary reason to use cloud-native systems might be the performance for some. At the same time, security cannot be omitted. However, there are many other aspects to this not mentioned directly, such as maintainability, which drives the whole category of evolution.

Finally, many works [1,14,63–65] have performed survey on the state of the art on related topics. Table 6 summarizes specific goals assessed in this study.

Table 6. Works relevant to RQ2.

Goal	References	Total
Microservice migration	[15–21,23,24,32,33,48–51,56,64]	17
Technical debt identification	[29,39,63]	3
SAR	[25,26,34,57]	4
Architectural analysis	[1,14,28,47,63]	5
Quality attribute analysis	[12,30,35–40,58–60]	11
Evolution analysis	[2,22,26–29,31,34,40,57,63,65]	12
Fault analysis	[2,30,38,41–47,54,55]	14
↪ *Root cause analysis*	[43,47,54,55]	↪ 4
Survey works	[1,14,45,63–65]	6

5.3. RQ3: Relationship between Microservices and Other Architectures

We found that microservices share more differences than similarities with other architectures. Works considering related architectures are referenced in Table 7. The most related architectural style is Service-Oriented Architectures (SOA) because microservice architectures borrow many characteristics from SOA, such as the emphasis on scalability and the concept of service availability and responsiveness. Unlike SOA, however, individual services in microservice architectures are brought to production independently and are more granular [63]. It is also worth noting that while graph-based methods of performing root cause analysis can be used in both microservice architectures and SOA [47], many traditional methods of code analysis are not sufficient for microservice architectures [14].

The greatest concern of much research towards static-code analysis is that microservices are distributed. Thus, it is rather common to observe static-code analysis in monolith-like systems since the system is homogeneous and often comes with one codebase. On the other hand, microservices, as opposed to monoliths, are heavily distributed and heterogeneous, and thus the argument might sound valid. However, still static-code analysis is applied to individual system modules; in addition, with sufficient cross-platform support, it is possible to derive a holistic view of system architecture of microservices solely by static-code analysis [26].

However, what seems lacking in nearly all works we assessed is consideration of other recent architectural advancements such as serverless or micro-frontends [64]. Of course, one can object that this study searched for microservices, but the argument by Auer et al. [64] remains valid: if researcher and practitioner do not understand the benefits of these new architectural advancements, they are possibly decreasing productivity or increase technical debt. This could be seen as similar to using SOA to develop a new system.

Table 7. Works relevant to RQ3.

Other Architecture	References	Total
Service-Oriented Architecture	[14,47,63]	3
Serverless and Micro-Frontends	[64]	1
Monoliths	[15,16,19–21,23,24,32,48–50,56,64]	13

5.4. RQ4: Future Research Directions

We have assessed the identified literature for future work and open challenges. This section mentions our observations on static analysis, dynamic analysis, anomaly detection, prediction, migration from monoliths to microservices, and other topics we found open for research, such as visualization, architecture evolution, and benchmarks.

The first significant conclusion we make is that all forms of static analysis seemed under-represented in the current literature. There is currently a greater focus on forms of dynamic analysis when analyzing microservices, leaving a gap for targeting statically defined artifacts. This kind of analysis can be done earlier in the development pipeline as the system does not need to be deployed for the analysis to take place, and it is less prone to false positives that may plague dynamic sources of information. However, the greatest challenge to address is to cope with distributed system nature and heterogeneity of system modules that may include distinct platforms, different versions and dependencies, or different development styles. Thus, future tools cannot just naively only consider Java as is the case for many works. Rather, a broad spectrum of languages such as Python, NodeJS, Go, C++, etc., must be considered to be well.

The large amount of research involving graph-based architectural analysis has opened up many new avenues for future research. One such avenue could be using graph-based static-code analysis to identify the potential for architectural degradation early on. Deriving the system into a graph brings needed abstraction. Such an approach could be beneficial when resolving degradation, which is difficult to prevent, or at least it could identify it early on.

With respect to dynamic analysis, we found multiple obstacles. There is considerable data collection overhead to collect metrics. In fact, a huge number of traces are produced at runtime, which makes it challenging to capture the required information in real time. In particular, the trace data need to be efficiently processed to produce aggregated trace representations of different levels of quality, and such detailed information of specific traces might need to be available on-demand. Even if we manage that, we need to store the data and analyze them. Thus, research must consider tracing microservices at a massive scale. This is rarely the case because of lacking benchmarks, which we mention later. Researchers questioned which metrics should be monitored and whether the metrics' accuracy or their impact on performance has been considered at large scales.

Dynamic analysis is often involved in anomaly detection and especially root cause analysis. In this context, Brandon et al. [47] highlighted the need for taking into account a time dimension, where the evolution of the system during a time window can be compared instead of single snapshots. However, what remains a challenge is the comparison between generated graphs representing the snapshots. This can be parallelized, but the search space and the response time need to be reduced for real-time processing. They proposed one way to address by transforming graphs into vectors to feed machine learning models dealing with anomaly detection, but this was just their vision for future work. Given machine learning, multiple works suggested the use of machine learning in this context, while statistical analysis can also be used for root cause analysis. We also found little relevant research on using machine learning as a primary method for microservice analysis. The only notable work that used machine learning as a primary method of analysis was written by Jin et al. [43], where a Robust Component Principal Analysis algorithm was used alongside dynamic analysis to detect anomalies. Zhou et al. [45] briefly mention the potential of using machine learning-based algorithms for improving fault localization. Other than the mentions in these articles, there were not any other notable instances of machine learning used as a primary method.

One outstanding challenge is prediction. For instance, since tuning configuration parameters can improve latency [36], researchers could look at the prediction of calls. Prediction of future system call could predict possible faults and learn from comparing the predicted system call and the actual to find specific patterns. Predictive models could help

to detect potential system bottlenecks and system capacity saturation for timely reactions to better handle such situations [12].

A trendy research topic is migration from monolithic applications to microservices. Since microservice-based applications have many desirable characteristics, a wide array of tools and methods that aid in the migration process would be extremely desirable. Further research into a way to accurately automate the process of decomposition with minimal impact by identifying potential candidate microservices would be of extreme significance for the industry. This would be beneficial because the current automated decomposition methods are either inaccurate or could have significant repercussions on the system, and human analysis is extremely time-consuming and sometimes leads to nowhere. A better-automated tool would allow companies to focus resources on further developing and improving their architecture.

One specific problem in microservice migration has many names and deals with accurate system decomposition, service boundaries, or proper service cuts. Some suggest that there are too many characteristics (e.g., non-functional requirements [50]) to take into account and propose the task for artificial intelligence. However, it can be a dynamic problem of continuous microservice system re-modularization. The situation is complicated by new advancements such as serverless or micro-frontends; should they be considered, or does migration from monoliths ultimately lead to technical debt? Perhaps the core challenge is understanding how to develop microservice systems. Alternatively, perhaps we need better evaluation metrics to answer this challenge.

One of the topics with great potential is software architecture reconstruction, especially its automation. Current approaches use static or dynamic analysis, but joint forces are inevitable since one deals well with decentralization and the other with a white-box view. Such system architecture can help with consistency checking and be the core artifact to refer to when dealing with evolution and technical debt. One challenge is proper visualization of the architecture or its perspectives.

In the context of visualization, researchers challenged proper execution trace visualization or improved fault localization. Others challenged visualization for data-flow across the system, recognizing reads and writes. Furthermore, it could be used for modeling and simulation of an actual microservices-based application.

With respect to technical debt, metrics for measuring debt are needed to quantify costs and benefits and support prioritization and decision-making. More investigation is needed on the relationship and composition between microservices availability tactics and microservices patterns. And some take the ambitious future goal to define an exhaustive and uniform catalog of microservices antipatterns.

Finally, one of the greatest deficiencies in related research is the lack of benchmarks [45]. We need more microservice data sets to test the systems. These specifically need to represent industrial settings [45]. Furthermore, to support advancements, unification could occur, and researchers should develop theme-specific benchmarks, such as a unified benchmark for fault injections and anomaly detection where approaches can compare easily, similar to what is common in other disciplines.

6. Threats to Validity

Mapping studies usually suffer from several threats to validity that need to be addressed. We tried to eliminate the effect of these threats on the quality of the results and the study's outcome. We discuss the validity threats from the perspective of Wohlin's taxonomy [69]. It includes four potential threats, i.e., external validity, construct validity, internal validity, and conclusions validity.

6.1. Construct Validity

Construct validity considers the investigated area with respect to the research questions. The primary term microservice and its immediate extension microservices used to

conduct this study are combined with secondary terms as specified in Section 3. All the primary and secondary terms are commonly recognized to be used as search strings.

A possible threat to the validity of our work is omitting relevant research from our review. We have tried to eliminate the effect of this threat by selecting and examining several search strings and conducting pilot searches for several papers. To ensure we found all related work from our initial search, we designed our query to be as broad as possible.

However, another perspective must be considered. Our study employed six major research databases, namely ACM Digital Library, IEEE Xplore, SpringerLink, ScienceDirect, MDPI and Wiley, although the authors had limited SpringerLink access to indexed full texts. Potentially, more papers can be indexed and published by other publishers, which we did not include. The analyzed sample only considered peer-reviewed articles published by journals or conferences to ensure the objectivity and reliability of the information sources. It does not include reprints of the papers submitted to or accepted in journals and conferences published by arXiv.org, researchgate.net, or individual personal pages. These reprints might contain novel ideas, methods, and new challenges relevant to the scope of analyzed papers.

6.2. Internal Validity

Internal validity challenges the methods employed to study and analyze data (e.g., the types of bias involved).

The *study search* perspective aims to assure we gathered all related papers on the selected topic. We searched through common publication databases indexing peer-reviewed literature (excluding gray literature). With regard to reliability and reproducibility, we defined search terms and applied procedures that others can replicate.

The next potential threat is related to the *paper's inclusion and exclusion* due to its scope. Due to the broad range of papers on microservices and the wide variety of research goals and aims, we have spent a lot of time scanning and reading the selected papers to ensure that the papers are within the scope of the study. The selection criteria are detailed in Section 3. For example, we excluded papers without specific output, giving suggestions or opinions regarding microservice analysis without experiments or robust proposed methods, or literature focused on analysis outside the scope of architecture, its design, or evolution. Other microservice-related questions were covered by several studies related to other microservice aspects, such as [3,5,9–11,70].

One potential threat is a *data extraction bias*. The source of this bias could be the extraction process when just one person extracts the information from the papers. To mitigate the effect of this threat, we distributed the data extraction among the different authors. Moreover, we had multiple authors review each search result and double-check other authors' extraction. Part of this process was the use of shared spreadsheets for rating and result verification. This included permissive analysis of the works' title, abstract, and keywords, followed by a more in-depth examination of the full text with extracts and categorization.

To address concerns related to *categorization/classification bias*, we have developed a mind map circulated across all authors for discussion, comments, and extension. The categorization is not exclusive and cannot be. It represents our view on the identified literature, and other alternative categorizations could be developed given the overlapping perspectives.

Data synthesis bias may affect the interpretation of the results. To mitigate this threat, the synthesis of the collected data was performed by multiple authors in multiple iterations with review sessions.

6.3. External Validity

External validity concerns knowledge generalization. In this study, we collected information to form a large scope of online sources. Our results and observations apply to microservices. However, they might be partially applicable to other system architectures. We analyzed and categorized given works based on the scope of research within the field or microservices, and thus our categorization cannot be implicitly generalized.

In addition, there is a risk of potentially impacting generalization within the microservices field, with possibly missing related work. In particular, our observations are a result of a peer-reviewed literature search published between 2018–2021 at 6 indexing sites.

6.4. Conclusions Validity

Conclusions validity concerns whether the conclusions are based on the available data. We had multiple authors involved in addressing this threat, double-checking the publication's rating and extracts to limit author bias, extraction bias, and interpretation bias. The conclusions result from several brainstorming sessions. Moreover, they were independently settled by all authors.

7. Conclusions

Microservices present a unique challenge for analysis due to their decentralized, independent nature. Due to this, a variety of unique methods have been developed to extract information from these independent services for a variety of end goals. In this study, we have performed a systematic mapping study to assess the current state of the recently published literature regarding methods of microservice analysis.

It first examined the works to discover and categorize five approaches and techniques that have been applied for microservice analysis. Next, it identified the problems and challenges these approaches were being used to address into seven categories with an additional category for surveys. Finally, it discussed the relationship between current and advancing architectures and the potential for future research directions in this area.

This study contributes to an understanding of the current literature surrounding microservice analysis. In particular, the identified most common methodologies currently in use include graph-based analysis and dynamic analysis. It also identified the most common goals of such analysis to be applied primarily to fault analysis and several others.

Although some might think that microservice architecture is well-established with well-known properties and design practice, it is still a challenge to properly divide the system into microservices, and broad research suggests various migration strategies but still admits there is space for improvements.

This study has shown that the influence and impact of microservice architecture have expanded greatly since its inception. Earlier studies have identified issues relating to the design and development of microservices, and our study shows that interest in this architectural style has expanded beyond into methods of structural analysis of microservice-based systems. The field of microservice analysis is still open to new innovations and methodologies, with a high potential reward for a more robust support environment for microservices.

Our original intent is to address the issues of centralized architectural design and architectural degradation. The first is addressed by various attempts, mostly leading into a software architecture reconstruction using graph-based or model-based approaches while combining static and dynamic analysis to extract information. The other spins around the same architectural view but adds patterns, especially antipatterns, that indicate a bad smell in the design, possibly leading to technical debt or degradation.

One valuable outcome of this study is the roadmap into the research that can serve researchers and an index to advancement or practitioners on assessing what is yet accomplished and which tools can be used. Still, many remaining challenges were identified for further advancement in the discipline. However, we believe that a robust architecture reconstruction is one of the core goals that need more research to better face system understanding, centralized perspectives, and system evolution and to avoid degradation.

The limitations of this work were discussed in a section on threats to validity, and certainly, a broader picture could be given in the future to address other architectures such as peer-to-peer systems or serverless.

Author Contributions: Conceptualization, V.B., T.C., D.D., A.A.M. and A.S.A.; methodology, V.B., D.D. and T.C.; validation, V.B., K.F., P.T. and M.B.; investigation, V.B., A.S.A., A.A.M., D.D., E.J. and M.C.; resources, V.B., A.S.A., A.A.M., D.D., A.L., E.J. and M.C.; data curation, V.B., A.S.A., A.A.M., D.D., A.L., E.J., M.C. and T.C.; writing—original draft preparation, V.B., A.L., E.J. and M.C.; writing—review and editing, V.B., T.C., A.S.A., A.A.M. and D.D.; visualization, V.B. and T.C.; supervision, V.B. and T.C.; project administration, T.C.; funding acquisition, T.C. All authors have read and agreed to the published version of the manuscript.

Funding: This material is based upon work supported by the National Science Foundation under Grant No. 1854049 and a grant from Red Hat Research https://research.redhat.com.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cerny, T.; Donahoo, M.J.; Trnka, M. Contextual Understanding of Microservice Architecture: Current and Future Directions. *SIGAPP Appl. Comput. Rev.* **2018**, *17*, 29–45. [CrossRef]
2. Ma, S.P.; Fan, C.Y.; Chuang, Y.; Liu, I.H.; Lan, C.W. Graph-based and scenario-driven microservice analysis, retrieval, and testing. *Future Gener. Comput. Syst.* **2019**, *100*, 724–735. [CrossRef]
3. Soldani, J.; Tamburri, D.A.; Van Den Heuvel, W.J. The pains and gains of microservices: A systematic grey literature review. *J. Syst. Softw.* **2018**, *146*, 215–232. [CrossRef]
4. Alshuqayran, N.; Ali, N.; Evans, R. A systematic mapping study in microservice architecture. In Proceedings of the 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 4–6 November 2016; pp. 44–51.
5. Pahl, C.; Jamshidi, P. Microservices: A Systematic Mapping Study. In Proceedings of the 6th International Conference on Cloud Computing and Services Science, Rome, Italy, 23–25 April 2016; pp. 137–146.
6. Pease, A.; Niles, I.; Li, J. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In Proceedings of the Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web, Edmonton, AB, Canada, 28–29 July 2002; Volume 28, pp. 7–10.
7. Di Francesco, P.; Malavolta, I.; Lago, P. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA), Gothenburg, Sweden, 3–7 April 2017; pp. 21–30.
8. Di Francesco, P.; Lago, P.; Malavolta, I. Architecting with microservices: A systematic mapping study. *J. Syst. Softw.* **2019**, *150*, 77–97. [CrossRef]
9. Taibi, D.; Lenarduzzi, V.; Pahl, C. Architectural Patterns for Microservices: A Systematic Mapping Study. In Proceedings of the 8th International Conference on Cloud Computing and Services Science, Madeira, Portugal, 19–21 March 2018; pp. 221–232.
10. Taibi, D.; Lenarduzzi, V.; Pahl, C. Continuous architecting with microservices and devops: A systematic mapping study. In *International Conference on Cloud Computing and Services Science*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 126–151.
11. Waseem, M.; Liang, P.; Shahin, M. A systematic mapping study on microservices architecture in devops. *J. Syst. Softw.* **2020**, *170*, 110798. [CrossRef]
12. Brondolin, R.; Santambrogio, M.D. A Black-Box Monitoring Approach to Measure Microservices Runtime Performance. *ACM Trans. Archit. Code Optim.* **2020**, *17*. [CrossRef]
13. Petersen, K.; Vakkalanka, S.; Kuzniarz, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Inf. Softw. Technol.* **2015**, *64*, 1–18. [CrossRef]
14. Cerny, T.; Svacina, J.; Das, D.; Bushong, V.; Bures, M.; Tisnovsky, P.; Frajtak, K.; Shin, D.; Huang, J. On Code Analysis Opportunities and Challenges for Enterprise Systems and Microservices. *IEEE Access* **2020**, *8*, 159449–159470. [CrossRef]
15. Baresi, L.; Garriga, M.; De Renzis, A. Microservices Identification Through Interface Analysis. In *Service-Oriented and Cloud Computing*; De Paoli, F., Schulte, S., Broch Johnsen, E., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 19–33.
16. Saidani, I.; Ouni, A.; Mkaouer, M.W.; Saied, A. Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search. In *Service-Oriented Computing*; Yangui, S., Bouassida Rodriguez, I., Drira, K., Tari, Z., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 58–63.
17. Kamimura, M.; Yano, K.; Hatano, T.; Matsuo, A. Extracting candidates of microservices from monolithic application code. In Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference (APSEC), Nara, Japan, 4–7 December 2018; pp. 571–580.
18. Furda, A.; Fidge, C.; Barros, A. A practical approach for detecting multi-tenancy data interference. *Sci. Comput. Program.* **2018**, *163*, 160–173. [CrossRef]

19. De Alwis, A.A.C.; Barros, A.; Fidge, C.; Polyvyanyy, A. Availability and Scalability Optimized Microservice Discovery from Enterprise Systems. In *On the Move to Meaningful Internet Systems: OTM 2019 Conferences*; Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 496–514.
20. De Alwis, A.A.C.; Barros, A.; Fidge, C.; Polyvyanyy, A. Microservice Remodularisation of Monolithic Enterprise Systems for Embedding in Industrial IoT Networks. In *Advanced Information Systems Engineering*; La Rosa, M., Sadiq, S., Teniente, E., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 432–448.
21. Matias, T.; Correia, F.F.; Fritzsch, J.; Bogner, J.; Ferreira, H.S.; Restivo, A. Determining Microservice Boundaries: A Case Study Using Static and Dynamic Software Analysis. In *Software Architecture*; Jansen, A., Malavolta, I., Muccini, H., Ozkaya, I., Zimmermann, O., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 315–332.
22. Soldani, J.; Muntoni, G.; Neri, D.; Brogi, A. The μTOSCA toolchain: Mining, analyzing, and refactoring microservice-based architectures. *Softw. Pract. Exp.* **2021**, *51*, 1591–1621. [CrossRef]
23. Eski, S.; Buzluca, F. An Automatic Extraction Approach: Transition to Microservices Architecture from Monolithic Application. In Proceedings of the 19th International Conference on Agile Software Development: Companion, Porto, Portugal, 21–25 May 2018; Association for Computing Machinery: New York, NY, USA, 2018. [CrossRef]
24. Ren, Z.; Wang, W.; Wu, G.; Gao, C.; Chen, W.; Wei, J.; Huang, T. Migrating Web Applications from Monolithic Structure to Microservices Architecture. In Proceedings of the Tenth Asia-Pacific Symposium on Internetware, Beijing, China, 16 September 2018; Association for Computing Machinery: New York, NY, USA, 2018. [CrossRef]
25. Ma, S.P.; Fan, C.Y.; Chuang, Y.; Lee, W.T.; Lee, S.J.; Hsueh, N.L. Using service dependency graph to analyze and test microservices. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; Volume 2, pp. 81–86.
26. Walker, A.; Laird, I.; Cerny, T. On Automatic Software Architecture Reconstruction of Microservice Applications. In *Information Science and Applications*; Kim, H., Kim, K.J., Park, S., Eds.; Springer: Singapore, 2021; pp. 223–234.
27. Walker, A.; Das, D.; Cerny, T. Automated Microservice Code-Smell Detection. In *Information Science and Applications*; Kim, H., Kim, K.J., Park, S., Eds.; Springer: Singapore, 2021; pp. 211–221.
28. Walker, A.; Das, D.; Cerny, T. Automated Code-Smell Detection in Microservices Through Static Analysis: A Case Study. *Appl. Sci.* **2020**, *10*, 7800. [CrossRef]
29. Pigazzini, I.; Fontana, F.A.; Lenarduzzi, V.; Taibi, D. Towards Microservice Smells Detection. In Proceedings of the 3rd International Conference on Technical Debt, Xiamen, China, 28 June 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 92–97. [CrossRef]
30. Márquez, G.; Astudillo, H. Identifying Availability Tactics to Support Security Architectural Design of Microservice-Based Systems. In *Proceedings of the 13th European Conference on Software Architecture—Volume 2*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 123–129. [CrossRef]
31. Tighilt, R.; Abdellatif, M.; Moha, N.; Mili, H.; Boussaidi, G.E.; Privat, J.; Guéhéneuc, Y.G. On the Study of Microservices Antipatterns: A Catalog Proposal. In Proceedings of the European Conference on Pattern Languages of Programs, Online, 1–4 July 2020; Association for Computing Machinery: New York, NY, USA, 2020. [CrossRef]
32. Kecskemeti, G.; Kertesz, A.; Marosi, A.C. Towards a Methodology to Form Microservices from Monolithic Ones. In *Euro-Par 2016: Parallel Processing Workshops*; Desprez, F., Dutot, P.F., Kaklamanis, C., Marchal, L., Molitorisz, K., Ricci, L., Scarano, V., Vega-Rodríguez, M.A., Varbanescu, A.L., Hunold, S., et al., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 284–295.
33. Jin, W.; Liu, T.; Zheng, Q.; Cui, D.; Cai, Y. Functionality-oriented microservice extraction based on execution trace clustering. In Proceedings of the 2018 IEEE International Conference on Web Services (ICWS), San Francisco, CA, USA, 2–7 July 2018; pp. 211–218.
34. Kleehaus, M.; Uludağ, Ö.; Schäfer, P.; Matthes, F. MICROLYZE: A Framework for Recovering the Software Architecture in Microservice-Based Environments. In *Information Systems in the Big Data Era*; Mendling, J., Mouratidis, H., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 148–162.
35. Jiang, Y.; Zhang, N.; Ren, Z. Research on Intelligent Monitoring Scheme for Microservice Application Systems. In Proceedings of the 2020 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS), Vientiane, Laos, 11–12 January 2020; pp. 791–794. [CrossRef]
36. Somashekar, G.; Gandhi, A. Towards Optimal Configuration of Microservices. In Proceedings of the 1st Workshop on Machine Learning and Systems, Online, 26 April 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 7–14. [CrossRef]
37. Ravichandiran, R.; Bannazadeh, H.; Leon-Garcia, A. Anomaly detection using resource behaviour analysis for autoscaling systems. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 25–29 June 2018; pp. 192–196.
38. Chen, H.; Chen, P.; Yu, G. A Framework of Virtual War Room and Matrix Sketch-Based Streaming Anomaly Detection for Microservice Systems. *IEEE Access* **2020**, *8*, 43413–43426. [CrossRef]
39. Bogner, J.; Schlinger, S.; Wagner, S.; Zimmermann, A. A Modular Approach to Calculate Service-Based Maintainability Metrics from Runtime Data of Microservices. In *Product-Focused Software Process Improvement*; Franch, X., Männistö, T., Martínez-Fernández, S., Eds.; Springer International Publishing: Cham, Swizterland, 2019; pp. 489–496.

40. Samardžić, M.; Šajina, R.; Tanković, N.; Grbac, T.G. Microservice Performance Degradation Correlation. In Proceedings of the 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 28 September–2 October 2020; pp. 1623–1626. [CrossRef]
41. Du, Q.; Xie, T.; He, Y. Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring. In *Algorithms and Architectures for Parallel Processing*; Vaidya, J., Li, J., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 560–572.
42. Meng, L.; Sun, Y.; Zhang, S. Midiag: A Sequential Trace-Based Fault Diagnosis Framework for Microservices. In *Services Computing—SCC 2020*; Wang, Q., Xia, Y., Seshadri, S., Zhang, L.J., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 137–144.
43. Jin, M.; Lv, A.; Zhu, Y.; Wen, Z.; Zhong, Y.; Zhao, Z.; Wu, J.; Li, H.; He, H.; Chen, F. An Anomaly Detection Algorithm for Microservice Architecture Based on Robust Principal Component Analysis. *IEEE Access* **2020**, *8*, 226397–226408. [CrossRef]
44. Zuo, Y.; Wu, Y.; Min, G.; Huang, C.; Pei, K. An Intelligent Anomaly Detection Scheme for Micro-Services Architectures With Temporal and Spatial Data Analysis. *IEEE Trans. Cogn. Commun. Netw.* **2020**, *6*, 548–561. [CrossRef]
45. Zhou, X.; Peng, X.; Xie, T.; Sun, J.; Ji, C.; Li, W.; Ding, D. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IEEE Trans. Softw. Eng.* **2021**, *47*, 243–260. [CrossRef]
46. Guo, X.; Peng, X.; Wang, H.; Li, W.; Jiang, H.; Ding, D.; Xie, T.; Su, L. Graph-Based Trace Analysis for Microservice Architecture Understanding and Problem Diagnosis. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Online, 8–13 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1387–1397. [CrossRef]
47. Brandón, Á.; Solé, M.; Huélamo, A.; Solans, D.; Pérez, M.S.; Muntés-Mulero, V. Graph-based root cause analysis for service-oriented and microservice architectures. *J. Syst. Softw.* **2020**, *159*, 110432. [CrossRef]
48. Christoforou, A.; Garriga, M.; Andreou, A.S.; Baresi, L. Supporting the Decision of Migrating to Microservices Through Multi-layer Fuzzy Cognitive Maps. In *Service-Oriented Computing*; Maximilien, M., Vallecillo, A., Wang, J., Oriol, M., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 471–480.
49. Nunes, L.; Santos, N.; Rito Silva, A. From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts. In *Software Architecture*; Bures, T., Duchien, L., Inverardi, P., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 37–52.
50. Li, S.; Zhang, H.; Jia, Z.; Li, Z.; Zhang, C.; Li, J.; Gao, Q.; Ge, J.; Shan, Z. A dataflow-driven approach to identifying microservices from monolithic applications. *J. Syst. Softw.* **2019**, *157*, 110380. [CrossRef]
51. Stojanovic, T.D.; Lazarevic, S.D.; Milic, M.; Antovic, I. Identifying microservices using structured system analysis. In Proceedings of the 2020 24th International Conference on Information Technology (IT), Zabljak, Montenegro, 18–22 February 2020; pp. 1–4.
52. Guan, Z.; Lin, J.; Chen, P. On Anomaly Detection and Root Cause Analysis of Microservice Systems. In *Service-Oriented Computing—ICSOC 2018 Workshops*; Liu, X., Mrissa, M., Zhang, L., Benslimane, D., Ghose, A., Wang, Z., Bucchiarone, A., Zhang, W., Zou, Y., Yu, Q., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 465–469.
53. Liu, H.; Zhang, J.; Shan, H.; Li, M.; Chen, Y.; He, X.; Li, X. JCallGraph: Tracing Microservices in Very Large Scale Container Cloud Platforms. In *Cloud Computing—CLOUD 2019*; Da Silva, D., Wang, Q., Zhang, L.J., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 287–302.
54. Meng, Y.; Zhang, S.; Sun, Y.; Zhang, R.; Hu, Z.; Zhang, Y.; Jia, C.; Wang, Z.; Pei, D. Localizing Failure Root Causes in a Microservice through Causality Inference. In Proceedings of the 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS), Hang Zhou, China, 15–17 June 2020; pp. 1–10. [CrossRef]
55. Wu, L.; Tordsson, J.; Elmroth, E.; Kao, O. MicroRCA: Root Cause Localization of Performance Issues in Microservices. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–9. [CrossRef]
56. Sun, C.A.; Wang, J.; Guo, J.; Wang, Z.; Duan, L. A Reconfigurable Microservice-Based Migration Technique for IoT Systems. In *Service-Oriented Computing—ICSOC 2019 Workshops*; Yangui, S., Bouguettaya, A., Xue, X., Faci, N., Gaaloul, W., Yu, Q., Zhou, Z., Hernandez, N., Nakagawa, E.Y., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 142–155.
57. Rademacher, F.; Sachweh, S.; Zündorf, A. A Modeling Method for Systematic Architecture Reconstruction of Microservice-Based Software Systems. In *Enterprise, Business-Process and Information Systems Modeling*; Nurcan, S., Reinhartz-Berger, I., Soffer, P., Zdravkovic, J., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 311–326.
58. Khazaei, H.; Barna, C.; Beigi-Mohammadi, N.; Litoiu, M. Efficiency Analysis of Provisioning Microservices. In Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Luxembourg, 12–15 December 2016; pp. 261–268. [CrossRef]
59. Chondamrongkul, N.; Sun, J.; Warren, I. Automated Security Analysis for Microservice Architecture. In Proceedings of the 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), Salvador, Brazil, 16–20 March 2020; pp. 79–82. [CrossRef]
60. Klinaku, F.; Bilgery, D.; Becker, S. The Applicability of Palladio for Assessing the Quality of Cloud-Based Microservice Architectures. In *Proceedings of the 13th European Conference on Software Architecture—Volume 2*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 34–37. [CrossRef]

61. Mendonca, N.C.; Aderaldo, C.M.; Camara, J.; Garlan, D. Model-Based Analysis of Microservice Resiliency Patterns. In Proceedings of the 2020 IEEE International Conference on Software Architecture (ICSA), Salvador, Brazil, 16–20 March 2020; pp. 114–124. [CrossRef]
62. McZara, J.; Kafle, S.; Shin, D. Modeling and Analysis of Dependencies between Microservices in DevSecOps. In Proceedings of the 2020 IEEE International Conference on Smart Cloud (SmartCloud), Washington, DC, USA, 6–8 November 2020; pp. 140–147. [CrossRef]
63. de Toledo, S.S.; Martini, A.; Sjøberg, D.I. Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. *J. Syst. Softw.* **2021**, *177*, 110968. [CrossRef]
64. Auer, F.; Lenarduzzi, V.; Felderer, M.; Taibi, D. From monolithic systems to Microservices: An assessment framework. *Inf. Softw. Technol.* **2021**, *137*, 106600. [CrossRef]
65. Bogner, J.; Fritzsch, J.; Wagner, S.; Zimmermann, A. Limiting Technical Debt with Maintainability Assurance: An Industry Survey on Used Techniques and Differences with Service- and Microservice-Based Systems. In Proceedings of the 2018 International Conference on Technical Debt, Gothenburg, Sweden, 27 May–3 June 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 125–133. [CrossRef]
66. Kobayashi, K.; Kamimura, M.; Kato, K.; Yano, K.; Matsuo, A. Feature-gathering dependency-based software clustering using dedication and modularity. In Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM), Trento, Italy, 23–28 September 2012; pp. 462–471.
67. Lin, T.; Park, B.; Bannazadeh, H.; Leon-Garcia, A. Savi testbed architecture and federation. In *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 3–10.
68. Palladio. Available online: https://www.palladio-simulator.com (accessed on 16 August 2021).
69. Wohlin, C. *Experimentation in Software Engineering: An Introduction*; International Series in Engineering and Computer Science; Kluwer Academic: Boston, MA, USA, 2000.
70. Bogner, J.; Fritzsch, J.; Wagner, S.; Zimmermann, A. Industry practices and challenges for the evolvability assurance of microservices. *Empir. Softw. Eng.* **2021**, *26*, 104. [CrossRef]