applied sciences

MDPI

Systematic Review

Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review

Mehmet Söylemez [1], Bedir Tekinerdogan [2,*] and Ayça Kolukısa Tarhan [1]

[1] Department of Computer Engineering, Hacettepe University, 06800 Ankara, Turkey; mehmetsoylemez@hacettepe.edu.tr (M.S.); atarhan@hacettepe.edu.tr (A.K.T.)
[2] Information Technology Group, Wageningen University & Research, 6706 PB Wageningen, The Netherlands
* Correspondence: bedir.tekinerdogan@wur.nl

Abstract: Microservice architecture (MSA) is an architectural style for distributed software systems, which promotes the use of fine-grained services with their own lifecycles. Several benefits of MSA have been reported in the literature, including increased modularity, flexible configuration, easier development, easier maintenance, and increased productivity. On the other hand, the adoption of MSA for a specific software system is not trivial and a number of challenges have been reported in the literature. These challenges should be evaluated carefully concerning project requirements before successful MSA adoption. Unfortunately, there has been no attempt to systematically review and categorize these challenges and the potential solution directions. This article aims at identifying the state of the art of MSA and describing the challenges in applying MSA together with the identified solution directions. A systematic literature review (SLR) is performed using the published literature since the introduction of MSA in 2014. Overall, 3842 papers were discovered using a well-planned review protocol, and 85 of them were selected as primary studies and analyzed regarding research questions. Nine basic categories of challenges were identified and detailed into 40 sub-categories, for which potential solution directions were explored. MSA seems feasible, but the identified challenges could impede the expected benefits when not taken into account. This study identifies and synthesizes the reported challenges and solution directions, but further research on these directions is needed to leverage the successful MSA adoption.

Keywords: microservice architecture; distributed architecture; systematic literature review

check for updates

Citation: Söylemez, M.; Tekinerdogan, B.; Kolukısa Tarhan, A. Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. *Appl. Sci.* **2022**, *12*, 5507. https://doi.org/10.3390/app12115507

Academic Editor: Vito Conforti

Received: 18 March 2022
Accepted: 24 May 2022
Published: 29 May 2022

1. Introduction

Distributed software systems have come to the fore in the last decade with increased demand for Internet of Things applications and advancements in cloud infrastructure. As a response to this demand and following the developments of service-oriented architecture (SOA) [1], microservice architecture (MSA) has emerged as an architectural style by promoting the use of fine-grained services and the cooperation of the nodes in the cloud. With a set of principles and patterns that address many best practices such as service discovery, API gateway, and circuit breaker, MSA is aimed especially at the problems of availability, fault tolerance, scalability, and maintainability that are experienced in software applications employing SOA or monolithic style.

With the MSA style, a distributed software application can be structured as a set of small services that run in their own process and communicate using lightweight mechanisms [2]. These services are small, autonomous, and work together [3]. An important aspect that emerged with MSA is the autonomy of the services. It is very critical that the services are designed as fine-grained. At this point, domain-driven design (DDD) [4,5] principles could be applied to decompose domains into subdomains and identify bounded contexts.

MSA promises software development firms increased agility because it is more open to changing requirements and related use cases and technologies than monolithic applications [2]. Design, development, and infrastructure automation processes can be handled successfully with MSA. Infrastructure automation decreases manual effort in building, deploying, and operating microservices. On the other hand, decentralized governance and data management allow services to be independent [6]. Thanks to important benefits, several important vendors such as Amazon, Netflix, LinkedIn, and Spotify have implemented their applications using MSA [2,6,7].

Despite these identified advantages, it is not straightforward neither for software teams to use MSA in distributed projects nor for researchers to guide the teams in their successful MSA adoption. One of the basic reasons for this difficulty is that the concept of MSA is complex in terms of distributed service identification, management, and maintenance and, therefore, requires considering a number of challenges together with the related solutions. Successful adoption of MSA thus requires a clear insight into the challenges and the corresponding solutions. Unfortunately, so far, no study has provided a systematic review to categorize the MSA challenges and the potential solution directions.

This article aims to identify the state of the art of MSA and describe the challenges in its application. In addition, solution directions for the identified challenges are investigated. A systematic literature review (SLR) [8] has been chosen as the research method while carrying out the literature analysis, and a multiphase study selection process has been applied using the published studies since the introduction of MSA in 2014. Overall, 3842 papers discovered using a well-planned review protocol have been examined, and 85 of them have been identified as primary studies for further review concerning research questions. The SLR has resulted in the identification of nine basic categories and forty sub-categories of challenges and the corresponding solution directions that can be used to depict the state-of-the-art in MSA and provide a vision for further research.

The article is organized as follows. Section 2 provides basic concepts about microservice architecture. Section 3 describes the adopted research method. Section 4 provides an overview of the selected studies. Section 5 presents the results and identified challenges according to the research questions. Section 6 provides a discussion of the results. Section 7 presents the related studies. Finally, Section 8 concludes the paper.

2. Microservice Architecture

Microservices are autonomous services that are smaller and more convenient to work with compared to other architectural styles [6]. The autonomy of the services refers to the ability to change and be deployed independently. Microservices are structured around business logic and can be developed, deployed, tested, and operated independently, are often operated by independent teams. Furthermore, microservices can be written in different programming languages and use different technologies [6].

MSA consists of crucial building blocks such as main business services, infrastructural services, discovery mechanisms, and communication infrastructure [9]. Each block is isolated from other blocks and communicates with them using a lightweight protocol. Therefore, they are easy to evolve in time with respect to the needs of business or technology.

Microservices are to some extent considered a natural extension of SOA, which gives importance to autonomous and lightweight services [10]. It extends some points in SOA based on a few principles. First, microservices are organized around business capabilities and this makes it easy to identify service and team boundaries. Second, infrastructure automation plays a critical role for MSA. It is directly related to the continuous delivery and continuous integration pipeline. Developers must always be convinced that the application works properly so automated test and deployment processes must be defined and implemented. Third, MSA allows the system to grow in a natural way. This is achieved by applying iterative and incremental approaches to satisfy business requirements. Thus, applications continuously change and optimize.

Microservices change the way software is designed from start to finish. Therefore, microservices conform to the evolutionary design in which the business predicts that certain functions may fail in the future. Scalable business models need applications that can be restructured and increased as scenarios evolve. Since every microservice is a small business process and represents a small aspect of business functionality, it is easy to change the workflow [11]. Accordingly, MSA development is the result of an evolutionary process. The process begins with identifying and then implementing microservices. Then, it continues with using DevOps practices and shaping the organization accordingly. The next step is to have a self-service, on-demand, and elastic infrastructure. Following this step, it is important to set up continuous integration and continuous deployment (CI and CD) pipeline with automation. With such an infrastructure, advanced deployment techniques can be set up and the firm becomes ready to use microservices [12].

MSA should be used in situations where its benefits outweigh its cost. While all functionalities are put into a single process in monolithic applications, each functionality set is placed in a separate service in the MSA. Hence, monolithic applications are scaled by replicating themselves on multiple servers. However, microservices are scaled by distributing across servers and replicating as needed. Microservices are not the best solution in any case [13]. Use cases and conditions should be evaluated well, and decisions should be made—this way, the greatest advantage is obtained from the microservices. In addition, since autonomous services can be deployed independently and benefit from the flexibility of the cloud and the rapid provision of resources, cloud-based platforms can be easily used in MSA development [5,14].

3. Research Methodology

This article aims to identify the state of the art of MSA and describe the challenges in applying MSA and the corresponding solution directions. To this end, a systematic literature review (SLR) or systematic review was applied following the guidelines by Kitchenham and Charters [8]. This is a well-known protocol for performing systematic literature reviews in the software engineering community. The basic activities of the review are shown in Figure 1. The SLR starts with defining the research questions followed by a definition of the search strategy and the identification of the study selection and elimination criteria. Subsequently, study quality assessment criteria are defined and the data extraction form is developed. Once these steps are ready, the data synthesis method is developed.

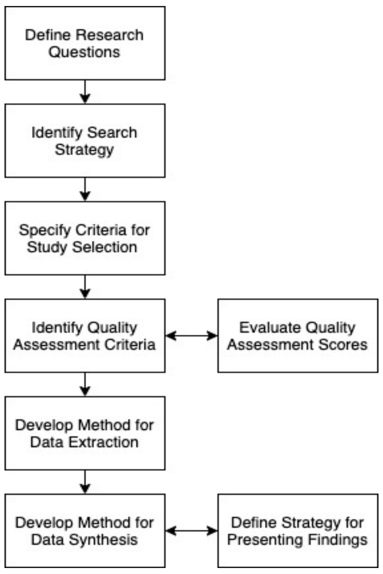
Define Research Questions
Identify Search Strategy
Specify Criteria for Study Selection
Identify Quality Assessment Criteria
Evaluate Quality Assessment Scores
Develop Method for Data Extraction
Develop Method for Data Synthesis
Define Strategy for Presenting Findings

Figure 1. Activities under the review protocol.

The research questions (RQs) of this SLR are given below:

RQ1. What are the identified challenges of microservice architectures?
RQ2. What are the proposed solution directions?

The studies published between January 2014 (which is the date when MSA was first defined by Lewis and Fowler [2]) and February 2022 were included in the SLR due to the fact that MSA was introduced in 2014. The electronic digital libraries included in the search were (in alphabetical order): ACM Digital Library, IEEE Xplore, Science Direct, Springer, and Wiley Inter-Science (see Table 1).

Table 1. Publication sources searched.

Source	# Studies Initially Retrieved	# Studies After Applying Exclusion/Quality Criteria
IEEE Xplore	233	48
ACM	755	12
Springer	1619	10
Science Direct	978	11
Wiley	174	4
Total	3842	85

Journal papers, conference papers, workshop papers, and books were considered potential search items. We used both automatic and manual search. The automatic search was performed by defining search strings using the APIs of the corresponding search databases. This was complemented with a manual search in which we used snowballing techniques. Hereby, the reference list of the selected primary studies is analyzed and the relevant studies are selected as primary studies (forward snowballing). On the other hand, we have looked at papers that cite the selected primary studies (backward snowballing). For selecting the primary studies, the following query was used:

((“micro service” OR “microservice”) AND (“challenge” OR “obstacle” OR “difficulty” OR “difficulties” OR “problem”))

We identified and used the exclusion criteria listed below to eliminate the studies that were irrelevant for this SLR:

EC 1: Studies with abstracts/titles that do not discuss MSA.
EC 2: Studies with abstracts/titles that do not bring an approach to MSA.
EC 3: Studies without a full text.
EC 4: Duplicate studies retrieved from different digital libraries.
EC 5: Studies that are not in English.
EC 6: Studies that do not explicitly discuss the challenges of MSA.
EC 7: Studies that relate to MSA but are experience and survey papers.
EC 8: Studies that present the application of MSA and do not critically reflect on MSA concepts.

The application of the exclusion and quality criteria eventually resulted in the selection of 85 papers from the initial set of 3842 papers.

The subsequent step included the quality assessment of the resulting primary studies, for which we used the quality checklists as defined in [15]. Accordingly, for the quality assessment, we used the checklist in Table 2. For the assessment scale we adopted a three-point scale, (i.e., yes = 1, somewhat = 0.5, no = 0). The scores for the assessment of the primary studies are provided in Appendix B. The authors performed a quality assessment. In case of conflicts, the results have been discussed, and a final conclusion has been provided.

Table 2. Quality assessment checklist.

No.	Assessment Question
Q1	Is the aim of the study defined clearly?
Q2	Are the scope, context and experimental design of the study stated clearly?
Q3	Does the study report have implications for research and/or practice?
Q4	Are the variables used in the study evaluation vali and reliable?
Q5	Are the measures used in the study explicit and aligned with the research purpose?
Q6	Is the research process documented in an understandable manner?
Q7	Are the main findings of the study presented clearly in terms of validity and reliability?
Q8	Is there an explicit statement of the limitations of the study?

The SLR followed with the detailed analysis and data extraction of the full text of the 85 primary studies. In this SLR, the quality assessment was considered as part of the data analysis, and as such the review data was kept in the same form. To develop the data extraction, several pilot primary studies were used and after a number of iterations, the final data extraction form was provided based on the consensus of the three authors.

In the final step of the SLR, the data synthesis, a qualitative and quantitative analysis, was independently performed on the data extracted from the primary studies. In addition, we discussed and selected suitable visual representations to support the synthesis process.

4. Overview of Selected Studies

The list of primary studies identified by this SLR is given in Appendix A. In Figure 2, we present the distribution of the primary studies by year. From the figure, we can observe a growing interest in the studies since 2015, following the year that MSA was proposed.

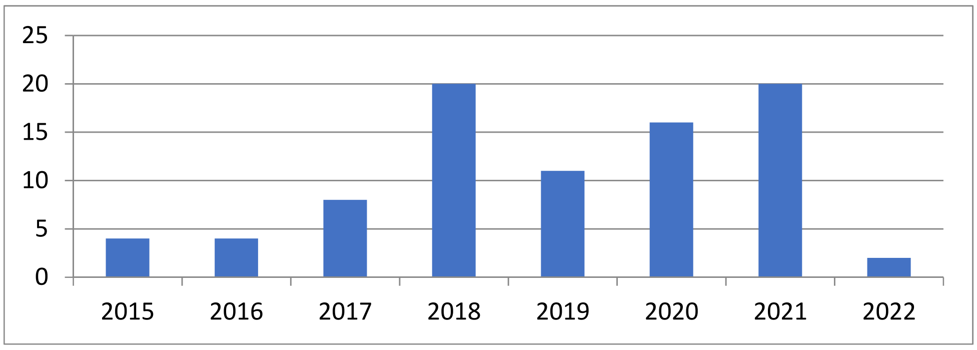

Figure 2. Year-wise distribution of the number of primary studies.

We have also analyzed the research methods employed in the primary studies to investigate the strength of evidence in these. Table 3 presents the adopted research methods in 85 primary studies. Table 4 presents the publication venues with the occurrence of primary studies. The research methods have been selected based on the SLR guidelines. As shown in the table, six different types of research methods were searched in the review. From the table, we can observe that most of the primary studies are based on a single case study. A summary of the venues where the primary studies were published is shown in Table 3.

Table 3. Studies by research methods.

Adopted Research Method	Study Labels	# Studies	Percentage
Descriptive or Not described	A, B, I, J, P, S, Y, AG, AH, AJ, AL, AQ, BE, BL	14	16.48%
Single-case	D, F, G, K, L, M, N, O, T, U, V, X, Z, AA, AB, AC, AD, AE, AF, AI, AM, AN, AP, AR, AT, AW, AX, AY, BF, BG, BH, BI, BJ, BK, BM, BN, BO, BQ, BR, BS, BT, BV, BW, BX, CA, CB, CC, CE, CG	49	57.64%
Multiple-case	C, E, H, Q, R, W, AK, AO, AS, AU, AV, AZ, BA, BB, BC, BD, BP, BU, BY, BZ, CD, CF	22	25.88%
Experiment	-	0	0%
Benchmarking	-	0	0%
Survey	-	0	0%

Table 4. Publication venues with the occurrence of primary studies.

No.	Publication Channel	Type	No. of Studies
1	ACM Symposium on Cloud Computing	Symposium	1
2	Annual International Conference on Computer Science and Software Engineering	Conference	1
3	Asian Simulation Conference	Conference	1
4	Conference on Innovations in Clouds, Internet and Networks (ICIN)	Conference	1
5	European Conference on Software Architecture	Conference	1
6	European Conference on Computer Systems	Conference	1
7	European Conference on the Engineering of Computer-Based Systems	Conference	1
8	Future Generation Software Systems	Journal	1
9	IEEE Annual Computer Software and Applications Conference (COMPSAC)	Conference	1
10	IEEE Cloud Computing	Journal	1
11	IEEE Transactions on Parallel and Distributed Systems	Journal	3
12	IEEE Conference on Energy Internet and Energy System Integration (EI2)	Conference	1
13	IEEE International Conference on Cloud Computing Technology and Science (CloudCom)	Conference	2
14	IEEE International Conference on Collaboration and Internet Computing (CIC)	Conference	1
15	IEEE International Conference on Communications (ICC)	Conference	1
16	IEEE International Conference on Distributed Computing Systems (ICDCS)	Conference	1
17	World Conference on Computing and Communication Technologies	Conference	1
18	IEEE International Conference on Software Architecture (ICSA)	Conference	1
19	IEEE Symposium on Service-Oriented System Engineering	Symposium	3
20	IEEE Conference on Computer Communications	Conference	1
21	IEEE Transactions on Services Computing	Journal	2
22	IEEE/ACM Symposium on Software Engineering in Africa (SeiA)	Symposium	1
23	International Conference in Software Engineering Research and Innovation (CONISOFT)	Conference	1
24	International Conference on Ambient Systems, Networks and Technologies	Conference	1
25	International Conference on Availability, Reliability and Security	Conference	1

Table 4. *Cont.*

No.	Publication Channel	Type	No. of Studies
26	International Conference on Information Integration and Web-based Applications and Services	Conference	1
27	International Conference on Information Technology in Bio- and Medical Informatics	Conference	1
28	International Conference on Performance Engineering	Conference	3
29	International Conference on Service-Oriented Computing	Conference	2
30	International Conference on Systems Science	Conference	1
31	International Conference on Ubiquitous and Future Networks (ICUFN)	Conference	1
32	International Conference on Web Research (ICWR)	Conference	1
33	International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)	Conference	1
34	International Symposium on Dependable Software Engineering: Theories, Tools, and Applications	Symposium	1
35	International Conference on Information Technology and Computer Application (ITCA)	Conference	1
36	International Symposium on Intelligent and Distributed Computing	Symposium	1
37	IEEE Internet of Things Journal	Journal	4
38	IEEE Cloud Summit	Conference	1
39	IEEE Access	Journal	4
40	IEEE Transactions on Cloud Computing	Journal	1
41	IEEE International Conference on Software Architecture Workshops	Conference	1
42	IEEE Transactions on Services Computing	Journal	1
43	Journal of Network and Computer Applications	Journal	2
44	Journal of Systems and Software	Journal	5
45	International Conference on Future Networks and Distributed Systems	Conference	1
46	Software: Practice and Experience	Journal	2
47	Microservices: Science and Engineering	Book	1
48	European Software Engineering Conference and Symposium on the Foundations of Software Engineering	Conference	1
49	Service Oriented Computing and Applications	Journal	1
50	IEEE International Enterprise Distributed Object Computing Workshop	Workshop	1
51	IEEE International Symposium on Network Computing and Applications	Conference	1
52	IEEE International Conference on Enterprise Distributed Object Computing (EDOC)	Conference	1
53	IEEE International Conference on Dependable Systems and Their Applications	Conference	1
54	IEEE/ACM International Symposium on Quality of Service	Conference	1
55	International Conference on Artificial Intelligence and Big Data	Conference	1
56	International Conference on Big Data Engineering and Education	Conference	1
57	International Conference on Software Engineering	Conference	1
58	IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing	Conference	1
59	Asia-Pacific Workshop on Networking	Workshop	1
60	Cognitive Computation and Systems	Journal	1
61	Concurrency and Computation: Practice and Experience	Journal	1
62	Journal of Systems Architecture	Journal	1
63	Future Generation Computer Systems	Journal	1
64	IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)	Workshop	1

The result of the quality assessment using the quality checklist in Table 2 is shown in Figure 3. We aimed to address methodological quality in terms of rigor, credibility and relevance, and reporting quality.

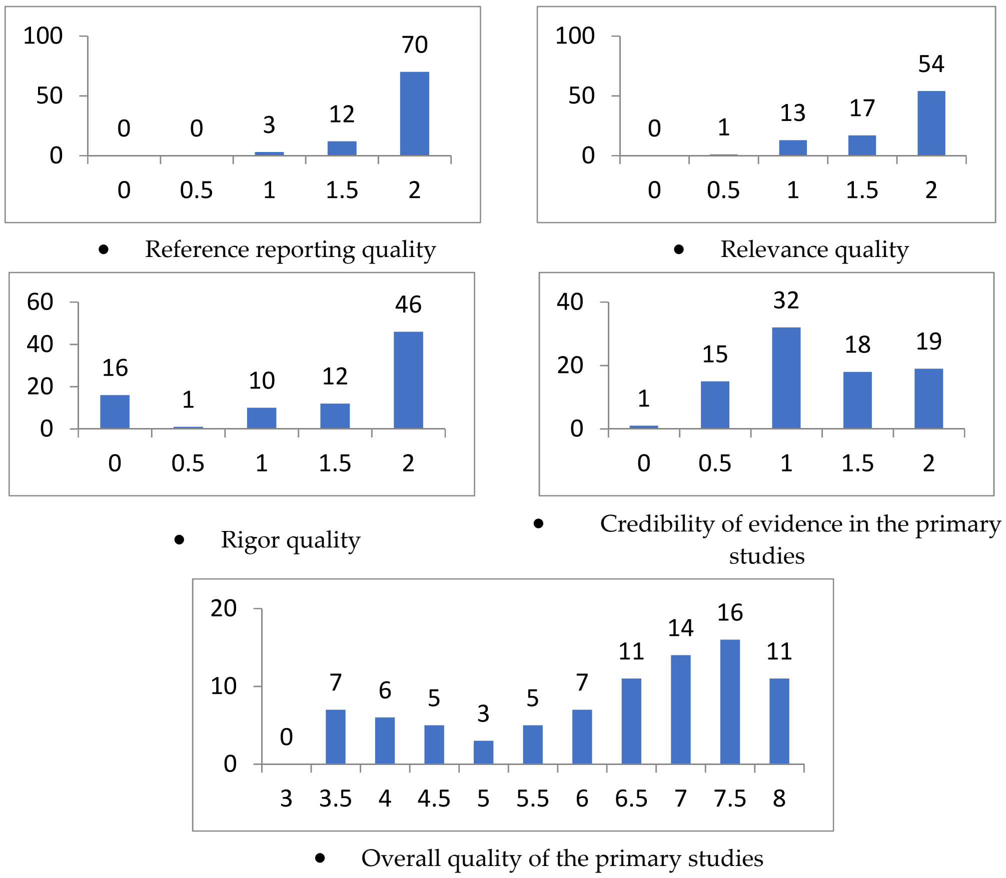

Figure 3. Quality metric results for the primary studies.

From this quality assessment, it was concluded that the majority of the primary studies (82.3%) are good with respect to reporting quality, and 63.5% of the studies (54 studies) were directly relevant to the field. Considering the rigor of the research methods, we can observe that 58 of the primary studies (68.2%) properly present the validity of their findings. In terms of rigor, forty-six studies demonstrate top quality. For the credibility of evidence, nineteen studies got the highest score with reasonably valid and meaningful findings and corresponding conclusions. As a result of the quality scores for reporting, relevance, rigor, and credibility of evidence, we can state that fifty-nine studies (69.4%) with scores equal to or greater than six are relatively good, eleven studies being high quality. On the other hand, twenty-six studies with scores less than six are identified as being of poor quality. As a result, most of the reviewed studies are assessed to be good.

5. Identified Challenges and Solution Directions

The results obtained in relation to the research questions are outlined in this section. The data extracted from the primary studies are summarized with findings separately for each question. The citations to the primary studies from which we have derived the findings are provided in Figure 4.

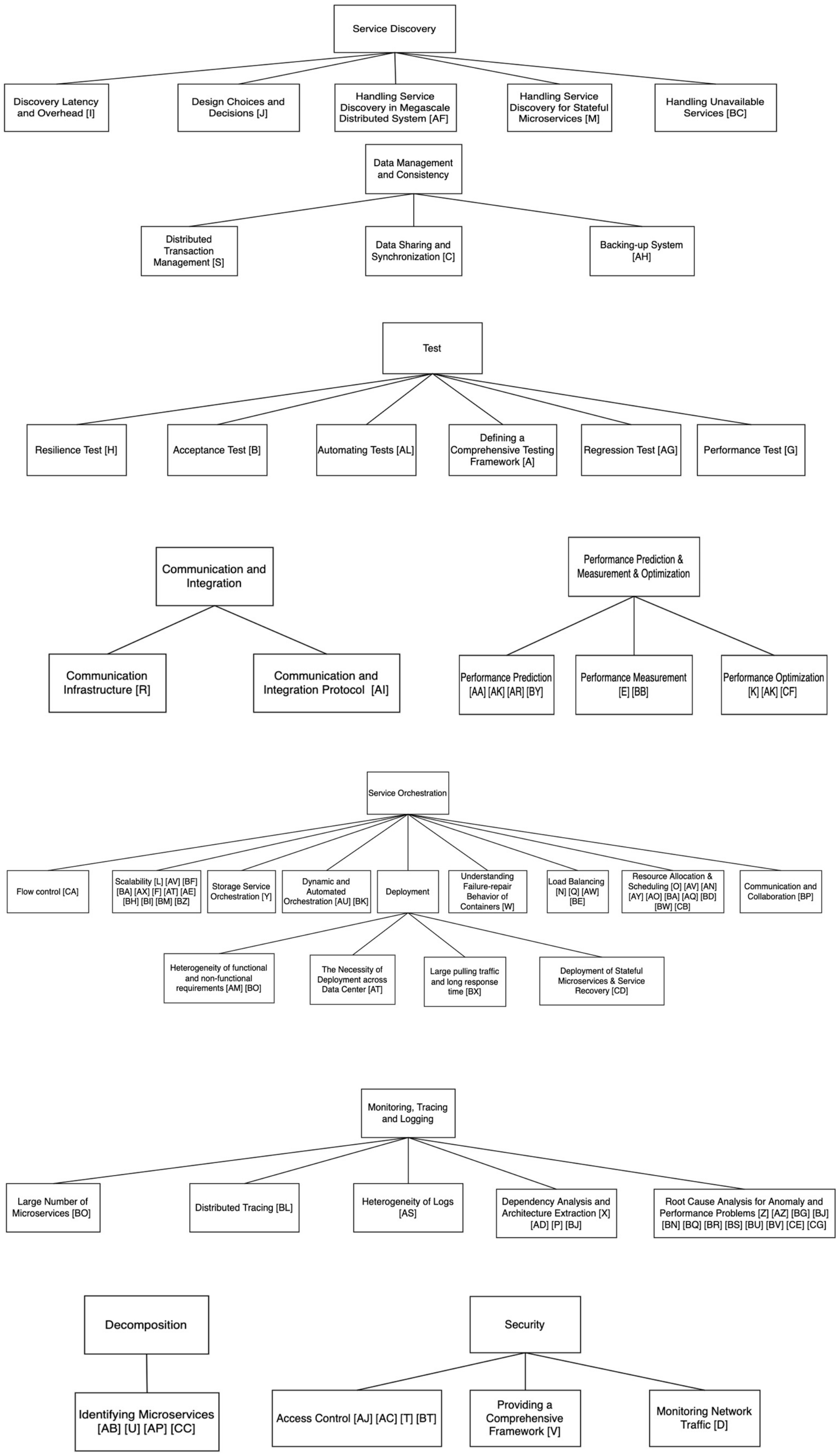
Service Discovery
Discovery Latency and Overhead [I]
Design Choices and Decisions [J]
Handling Service Discovery in Megascale Distributed System [AF]
Handling Service Discovery for Stateful Microservices [M]
Handling Unavailable Services [BC]
Data Management and Consistency
Distributed Transaction Management [S]
Data Sharing and Synchronization [C]
Backing-up System [AH]
Test
Resilience Test [H]
Acceptance Test [B]
Automating Tests [AL]
Defining a Comprehensive Testing Framework [A]
Regression Test [AG]
Performance Test [G]
Communication and Integration
Communication Infrastructure [R]
Communication and Integration Protocol [AI]
Performance Prediction & Measurement & Optimization
Performance Prediction [AA] [AK] [AR] [BY]
Performance Measurement [E] [BB]
Performance Optimization [K] [AK] [CF]
Service Orchestration
Flow control [CA]
Scalability [L] [AV] [BF] [BA] [AX] [F] [AT] [AE] [BH] [BI] [BM] [BZ]
Storage Service Orchestration [Y]
Dynamic and Automated Orchestration [AU] [BK]
Deployment
Understanding Failure-repair Behavior of Containers [W]
Load Balancing [N] [Q] [AW] [BE]
Resource Allocation & Scheduling [O] [AV] [AN] [AY] [AO] [BA] [AQ] [BD] [BW] [CB]
Communication and Collaboration [BP]
Heterogeneity of functional and non-functional requirements [AM] [BO]
The Necessity of Deployment across Data Center [AT]
Large pulling traffic and long response time [BX]
Deployment of Stateful Microservices & Service Recovery [CD]
Monitoring, Tracing and Logging
Large Number of Microservices [BO]
Distributed Tracing [BL]
Heterogeneity of Logs [AS]
Dependency Analysis and Architecture Extraction [X] [AD] [P] [BJ]
Root Cause Analysis for Anomaly and Performance Problems [Z] [AZ] [BG] [BJ] [BN] [BQ] [BR] [BS] [BU] [BV] [CE] [CG]
Decomposition
Identifying Microservices [AB] [U] [AP] [CC]
Security
Access Control [AJ] [AC] [T] [BT]
Providing a Comprehensive Framework [V]
Monitoring Network Traffic [D]

Figure 4. Visual summary of identified problems of MSA.

5.1. RQ1. What Are the Identified Challenges in the MSA Domain?

Table 5 shows an overview of the nine identified problems. The first column in the table presents the labels of the primary studies, the second column the publication date of the studies, and the remaining columns the identified problems (P1 to P9) in the studies. The description of the problems is shown at the right of the table. Figure 4 provides a visual summary of the identified challenges. Next, we discuss the challenges derived from the primary studies in the following subsections.

Table 5. Primary studies with identified problems of MSA.

P1	Service Discovery									
P2	Data Management and Consistency									
P3	Testing									
P4	Performance Prediction, Measurement and Optimization									
P5	Communication and Integration									
P6	Service Orchestration									
P7	Security									
P8	Monitoring, Tracing and Logging									
P9	Decomposition									
						Identified Challenges				
Study	**Year**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**
A	2015			X						
B	2015			X						
C	2015		X							
D	2015							X		
E	2016				X					
F	2016						X			
G	2016			X						
H	2016			X						
I	2017	X								
J	2017	X								
K	2017				X					
L	2017						X			
M	2017	X								
N	2017						X			
O	2017						X			
P	2017								X	
Q	2018						X			
R	2018					X				
S	2018		X							
T	2018							X		
U	2018									X
V	2018							X		
W	2018						X			

Table 5. *Cont.*

		Identified Challenges								
Study	**Year**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**
X	2018								X	
Y	2018						X			
Z	2018								X	
AA	2018				X					
AB	2018									X
AC	2018							X		
AD	2018								X	
AE	2018						X			
AF	2018	X								
AG	2018			X						
AH	2018		X							
AI	2018					X				
AJ	2018							X		
AK	2019				X					
AL	2019			X						
AM	2019						X			
AN	2019						X			
AO	2019						X			
AP	2019									X
AQ	2019						X			
AR	2019				X					
AS	2019								X	
AT	2019						X			
AU	2019						X			
AV	2020						X			
AW	2020						X			
AX	2020						X			
AY	2020						X			
AZ	2020								X	
BA	2020						X			
BB	2020				X					
BC	2020	X								
BD	2020						X			
BE	2020						X			
BF	2020						X			
BG	2020								X	
BH	2020						X			
BI	2020						X			
BJ	2020								X	

Table 5. *Cont.*

		Identified Challenges								
Study	**Year**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**
BK	2020						X			
BL	2021								X	
BM	2021						X			
BN	2021								X	
BO	2021								X	
BP	2021						X			
BQ	2021								X	
BR	2021								X	
BS	2021								X	
BT	2021							X		
BU	2021								X	
BV	2021								X	
BW	2021						X			
BX	2021						X			
BY	2021				X					
BZ	2021						X			
CA	2021						X			
CB	2021						X			
CC	2021									X
CD	2021						X			
CE	2021								X	
CF	2022				X					
CG	2022								X	
TOTAL: 85		**5**	**3**	**6**	**8**	**2**	**33**	**6**	**18**	**4**

5.1.1. Service Discovery

In a distributed architecture such as microservice architectures, the discovery of microservices is one of the primary challenges. The challenges for service discovery relate to the design, implementation, and quality concerns. At the design level, designing the service discovery is considered to be a challenge due to the multiple various service discovery mechanisms such as client-side, server-side, and hybrid service discovery. The proper decision needs to be made based on the various different requirements and quality concerns. Often multiple different design alternatives can be identified, and it is not easy to derive a feasible alternative. Implementing service discovery is directly dependent on system size and selected design so the most important criteria for the implementation are high availability and scalability. A misguided design selection and subsequent development will affect the system's availability and scalability. During the operation time or run-time, discovering the proper services requires the corresponding orchestration which needs to be aligned with the required quality of service parameters. One important quality factor is the latency of the discovered and triggered service.

5.1.2. Data Management and Consistency

Data management and consistency are challenges of MSA because of their distributed nature. The challenges related to data management and consistency are more about distributed transaction management, but it is also about backing up the system and data integration. Architects and developers often choose a database per service pattern to achieve distributed transaction management, and MSA also favors decentralized data management. Although this pattern comes with a lot of advantages such as loosely coupled services, and independently deployable and scalable services, the management of distributed transactions is really tough work. Backing up the entire application decomposed into microservices consists of some trade-offs, so it is not possible to handle backing up the entire system in addition to providing availability and consistency at the same time. Therefore, it is another challenging point for practitioners to decide which ones are more important according to the system design. In some MSAs, there is not a mature mechanism for data sharing and synchronization. Microservices can operate on each other's data without a coherent architecture, but then it makes the system more complex. It should be handled so that sharing and synchronization operations do not operate on each other's data.

5.1.3. Testing

Testing plays a critical role in a system being ready before going live, and for developers moving forward confidently. However, for MSA, it is a challenging task to satisfy testing activities due to the distributed nature of MSA. Each microservice lives in a distributed environment and can be developed using different technologies, languages, and infrastructures, which as such provides additional complexity for the testing process.

Testing the resilience capabilities of MSA is identified as an important challenge. Resilience enables systems to handle failure cases properly, but in a distributed architecture it is not easy to achieve this because microservices architecture is composed of a set of services that operate together and thus are prone to frequent changes. They should be loosely coupled and autonomous for being resilient as well.

Another challenging issue is performance tests. Non-functional requirements such as throughput and response time are important performance parameters for software systems. It is necessary to have performance tests to measure these kinds of performance parameters to trace the system properly and prevent any failure. In distributed environments, however, measuring these quality parameters is not as easy as in a monolithic architecture because of the diversity and number of microservices.

Regression testing is needed to ensure that the system is still running after a newly added feature or after a bug that has been resolved. Regression testing for MSA is not trivial, since all the test activities need to be handled in an agile way, they must be automated and included in the continuous delivery process.

Another issue is related to acceptance tests. They are a set of test activities that must be run to ensure customer satisfaction and create robust systems, but this requires high maintenance costs to be handled in the microservice world because of the agility of MSA.

In addition, defining a comprehensive testing framework has emerged as a tough work in recent years because it consists of many sub-challenging points such as self-validation of interfaces, unit validation, and integration validation of services. Additionally, each validation should be automated to provide continuous and agile deployments.

The last issue is automating tests. It is directly related to defining or reusing a framework to run tests automatically. The proper testing flow needs to be prepared and run from time to time depending on decisions on the test plan and at each testing cycle, it needs to ensure that it is not affecting the system's reliability, and it is not an easy work.

5.1.4. Performance Prediction, Measurement, and Optimization

Performance is a key quality factor for the systems employing MSA because it is addressed at different levels for the stages of system design, implementation and operation. Usually, it is beneficial to estimate the performance of a software system before it is implemented because it may be very difficult or costly to change the system afterward. After the implementation, performance measurement and optimization become important in order to satisfy the quality of MSA-based system requirements.

5.1.5. Communication and Integration

Communication and integration are other challenging points that have emerged as a result of distributed architecture. Even if microservices communicate with a more lightweight protocol, it is still difficult to ensure that the communication infrastructure is reliable and the protocol to be used for communication and integration can handle complex workflows. The most important criteria for both challenges are reliability and durability; if these criteria are not met, the proper operation and reliability of the system will be affected, and will possibly cause cascading failures in the system.

5.1.6. Service Orchestration

Service orchestration is a concept that contains deployment, scalability, scheduling, management, and networking of microservices. Although container orchestration tools that address many of these concepts have also been developed in industry, some studies suggest solutions for challenging points in each area. The challenges for service orchestration relate to scalability, dynamic and automated orchestration, storage service orchestration, deployment, load balancing, and scheduling. There are thus several sub-concerns related to this concern. The following concerns need to be considered from the context of service orchestration.

Adapting containers' resources dynamically according to the changing requirements makes MSA-based systems highly available. However, it is a challenging issue to trace the changing requirements of containers and make the necessary adjustments according to the usage of resources over time.

Providing persistent storage among different containers is another challenging point. With the increasing usage of deployment for stateful applications in the cloud, the need to address issues and challenges with persistent storage for containers has emerged. This problem includes many sub-challenging points such multi-protocol support and storage service orchestration for volume management. They need to be handled by designing a comprehensive solution to overcome workloads and resource problems.

Deployment is also a big challenge for practitioners. Although deployment processes gain a big momentum with containers, it still has some challenges. These challenges are mostly related to planning and configuring deployment. Additionally, decentralized deployments are newly emerged challenges due to the necessity of deploying across data centers. Finally, the heterogeneity of functional and non-functional requirements of microservices pushes practitioners to find an optimal deployment model to satisfy all these requirements for each service.

Load balancing plays a critical role in effectively distributing incoming requests to the backend servers. The most important criteria for load balancing are availability because requests that are not effectively distributed will cause the system to stop responding, which will affect system availability.

Auto-scalability is another challenging point for container orchestration because in distributed environments, services need to be monitored and automatically adjusted resources according to changing loads to keep applications predictable, resilient, and available. These requirements bring a lot of challenges for scalability.

Another challenging point is resource allocation and scheduling. It is an important activity to organize and manage the chain of services and schedule the available resources to effectively use. In addition, reducing total traffic cost and delay are important criteria for scheduling. Misguided scheduling directly affects the availability and reliability of the system.

Understanding failure–recovery behavior of containers is the last challenge of orchestration. It needs a comprehensive analysis of how the containers run from an availability and reliability point of view. Moreover, deployment configurations often need to be reviewed for effective analysis and if needed, should be improved.

5.1.7. Security

With the development of MSA and distributed architecture, security has started to be a more important topic because each microservice exposes a new entry point to both the internal and external sides. This situation brings with it a lot of security problems as long as it is not addressed.

The first problem is the access control mechanism in MSA. With the access control, the resource that is intended to be accessed is either restricted or not allowed. However, applying access control in the MSA is difficult to apply in a monolithic architecture because of its distributed nature.

Furthermore, building a comprehensive framework to establish security between microservice is tough work to achieve because it is very difficult to integrate and use unusable and non-lightweight frameworks comfortably.

Another important issue is the monitoring of the network traffic and running some security rules defined according to requirements.

5.1.8. Monitoring, Tracing, and Logging (MTL)

Monitoring, tracing, and logging are important activities to ensure that systems are able to satisfy availability, performance, and reliability concerns. However, these important activities consist of several challenging points related to identifying strong coupled services, the root cause of anomalies and performance problems, and the heterogeneity of logs.

The logs from different microservices might be heterogeneous so understanding and traversing the logs emerges as a challenging point. If the trace cannot be established among the logs, the ability to monitor the system is directly affected. Thus, practitioners will not make proper decisions for troubleshooting.

It is critical to identify these problems and take quick action as soon as possible. Otherwise, the system's availability, reliability, and fault tolerance will be directly affected.

Expected behavior for the MTL process is that trouble spots are detected, and the system is made more available, scalable, reliable, and fault-tolerant by taking quick actions or making changes in the design if necessary.

5.1.9. Decomposition

Decomposition allows us to have autonomous services organized around business capability services. it is necessary to separate the system into suitable pieces functionally and obtaining high cohesive and loosely coupled services are expected as a result of decomposition. The challenging point encountered first after deciding to go with MSA is to determine the right size of business capability and if it cannot succeed properly, MSA will not be an advantage and it might cause many problems in terms of mainly scalability, performance, availability and reliability.

5.2. RQ2. What Are the Identified Solution Directions?

When addressing the challenges of MSA, many studies provide the related solution directions together with the challenges. Table 6 summarizes the solution directions for the identified problems in Table 5. We observe in Table 5 that the solution directions are inherently varied based on the identified challenges. Solution directions include design heuristics and design abstractions, adoption of different paradigms, novel introduction and implementation of algorithms, integration with other paradigms, and solutions to realize system-wide quality management. Below, we will discuss the solution directions of each challenge separately.

Table 6. Solution directions for the identified challenges in MSA.

Primary Challenge	Solution Direction
P1. Service Discovery	- Client-Side Service Discovery. Study J - Server-Side Service Discovery. Study J - Service Registry. Study J - ICN-Based Service Discovery. Study I - Static–Dynamic Service Description. Study AF - Stateful Routing Mechanism. Study M
P2. Data Management and Consistency	- Multi Agent-Based Framework. Study S - BAC Theorem for Backing up Data. Study AH - Solution Framework. Study C
P3. Testing	- Reusable BDD-Based Acceptance Test Architecture. Study B - A Flow for Regression Test. Study AG - An Architecture and Framework to Automate Performance Test. Study G - A Framework for Testing the Failure-Handling Capabilities. Study H - Validation Framework. Study A - Automation Testing Framework. Study AL
P4. Performance Prediction, Measurement and Optimization	- Simulation Model. Study AA - Performance Model and Prediction Method. Study AK - Performance Prediction Model. Study AR - Performance Analytical Model. Study E - An approach for the Quantitative Assessment of Microservice Architecture Deployment Configuration Alternatives. Study BB - Performance Degradation Prediction Framework. Study BY - A Model-Driven Approach for Continuous Performance Improvement. Study CF
P5. Communication and Integration	- Reference Architecture and Orchestrator Language. Study AI - High-performance Userspace Networking Solution. Study R
P6. Service Orchestration	- An Extendable Solution for Autoscaling. Study L - Database-is-the-Service Pattern. Study F - Workflow Scheduling Algorithm. Study AV - Autoscaling Research Pipeline. Study BF - Ant Colony Algorithm for Microservice Scheduling Optimization. Study AN - A Novel Scheduling Strategy. Study BA - A Lightweight and Flexible System for Autoscaling. Study AX - A Generic Architecture and Implementation for Automated Orchestration. Study AU - Configuration Models and Tool for Analyzing the Availability. Study W - Storage Service Orchestrator Framework. Study Y - A Monitoring-Based Architecture for Managing Deployment. Study AM - Decentralized Orchestrator. Study AT - Process Definition for An Elasticity Controller. Study AE - Decentralized Load Balancing Algorithm. Study N - Overload Control Method. Study Q - A Hybrid Approach Combining Client-side and Server-side Load Balancing. Study BE - Queue-Based Chain-Oriented Load Balancing Method. Study AW - A Novel Fair Weighted Affinity-based Scheduling Approach. Study O - A Novel Scheduling Framework for Kubernetes. Study AQ - Dynamic Microservice Scheduling Algorithm for Mobile Edge Computing. Study AY - Resource Allocation Optimization Approach. Study AO - Many-Objective Genetic Algorithm Scheduler. Study BD - A Novel Formula and Model for Determining the Thresholds of Total Resource Consumption. Study BH - Autoscaling Research Pipeline. Study BI - Using Declarative Business Processes for Service Orchestration. Study BK - RL agent based intelligent autoscaling model. Study BM - A Decision Framework to Select Right Microservice Collaboration Pattern—Study BP - Elastic Scheduling Algorithm—Study BW - Layered Container Structure for Microservice Deployment—Study BX - Autoscaling Framework for Microservice chain—Study BZ - Dynamic Flow Control Algorithm—Study CA - Microservice Rescheduling Framework—Study CB - A Kubernetes Controller for Managing Availability—Study CD

Table 6. *Cont.*

Primary Challenge	Solution Direction
P7. Security	- An Approach That Provides Authentication and Decentralized Role-Based Authorization. Study T - A Platform for Identity and Access Control of Microservices. Study AC - Access Control Optimization Model. Study AJ - Prototype Layered Security Framework (Hardware, Virtualization, Cloud, Communication, Application, and Orchestration). Study V - An Approach for Handling Security as Security-as-a-Service. Study D - Extended Role-Based Access Control Model. Study BT
P8. Monitoring, Tracing and Logging	- An Approach and Tool for Generating Service Dependency Graph. Study X - A Tool for Generating Service Causal Graph. Study Z - An Approach For Analyzing Architecture. Study AD - A Tool For Handling Traversing Distinct Type of Logs. Study AS - A Tool For Architecture Recovery. Study P - A Root Cause Analysis Framework for Detecting Anomalies. Study AZ - An Execution Trace-Based Root Cause Location Method. Study BG - A Graph-based Trace Analysis Approach—Study BJ - An Offline Approach to Distributed Tracing—Study BL - A Four Layered Framework for Detection and Diagnosis of Faulty Microservices—Study BN - In-Kernel Transparent Monitoring Service—Study BO - Microservice Fault Detection Method Based on Correlation Analysis—Study BQ - A Fault Model-Based Root Cause Localization Framework—Study BR - An Anomaly Detection Method based on Semi-Supervised Learning—Study BS - A Root Cause Localization Approach—Study BU - Lightweight Spectrum-Based Performance Diagnosis Tool—Study BV - An Anomaly Detection Approach with Execution Trace Comparison—Study CE - An Agent-Based Monitoring Platform to Detect Anomalies and Unexpected System Dependencies—Study CG
P9. Decomposition	- A Conceptual Methodology for Deciding Right Size of Microservices. Study AB - A Functional Decomposition Approach. Study U - A Dataflow-Driven Decomposition. Study AP - A Dependency Capturing and Clustering-Based Microservice Identification Approach. Study CC

5.2.1. Service Discovery

The authors [I] propose a new approach that uses information-centric networking (ICN) to find a solution to latency and overhead problems of the service discovery mechanism. They make the service discovery process in MSA easier by using information-centric network concepts. Since it is possible to offer a simple discovery process that decreases the number of service name records, name-based routing, and hierarchical naming are used.

Study [J] presents multiple decision guidance models that can be used when deciding on an eligible microservice infrastructure. In this paper, there are multiple decision guidance models with their own design options addressing the fault tolerance and service discovery area. For each of the models, they provide specific infrastructure technologies to implement design options.

Study [AF] provides a novel solution architecture to solve scalability and workload problems of service discovery in mega-scale systems. The authors focus on the idea that service description data can be broken into dynamic and static properties. They propose an architecture subdivided into two independent, interconnected processing levels for static and dynamic query parts. Both processing levels consist of interconnected peers which allow scaling of the registry dynamically.

Study [M] proposes a mechanism to optimize service discovery operation in stateful microservices. Scalability and efficient usage of infrastructure resources are the main aspects of this study. The authors claim that an efficient and scalable routing mechanism is needed to figure these problems out. The proposed model has been validated with two experiments and it has been observed that there was an increase in scalability and a decrease in the usage of infrastructure resources.

Study [BC] argues that the unavailable states of services are useless and faulty interaction topics. The authors indicate that there should be a synchronization mechanism to support microservice communications. Hence, they offer a framework called Synchronizer. This framework achieves collecting health/state information of microservices by using distributed registries. This framework has been validated with multiple use cases and according to the result, it brings effectiveness to synchronization among microservices.

To sum up, the corresponding solutions to the service discovery are based mainly on scalability and workload problems. To cope with these problems, the identified primary studies propose either a decision model helping to choose the best option within the existing solutions or a novel model for handling service discovery problems.

5.2.2. Data Management and Consistency

Study [S] focuses on the management of distributed transactions. In this study, a multi-agent-based framework is proposed to coordinate distributed transactions of the system. This solution is based on agents associated with a particular microservice, eventual consistency, saga patterns [16], and a semi-orchestrated asynchronous model, and provides a decoupled autonomous layer to the application to simplify the microservice interactions.

Study [AH] introduces the backup, availability, and/or consistency (BAC) theorem to be used in backing up microservice. This theorem indicates that practitioners have to pick up two out of three items which are backup, availability, or consistency. This theorem, inspired by the CAP theorem [17], claims that it is not possible to satisfy both availability and consistency at the same time in backing up microservices.

Study [C] provides a Synapse framework supporting independent services to share data with each other through clean APIs. This study addresses the problem of complex service groups that do not have a consistent, manageable structure and operate on each other's data. Synapse provides a transparent data propagation layer by using the model–view–controller (MVC) framework and object/relational mappings (ORMs). Synapse has been implemented for Ruby on Rails and has shown that it provides good performance and scalability.

In sum, the identified primary studies indicate that it is difficult to manage data consistency, backup data, and data synchronization. For distributed transaction management, the identified solution proposes a multi-agent-based framework. For backing up, the identified solution proposes a novel BAC theorem. Finally, a solution comprehensive framework is proposed to be used in data synchronization.

5.2.3. Testing

In study [AG], the authors propose an automated method of running regression tests. They focus on the software reliability challenge. They claim that the regression test is an essential step in the continuous delivery process and in order to ensure reliability, the automated method of regression test plays a critical role. Authors define the process of how to run regression test automatically and place it in continuous delivery.

Study [B] presents reusable automated acceptance testing architecture to handle the maintainability and reusability of the application. This study encourages developers to use behavior-driven development (BDD) acceptance tests more frequently. The authors claim that this architecture will minimize issues with integration maintenance costs.

Study [A] presents an analysis of existing cloud application test methods and defines the characteristics of MSA. Based on this analysis, they propose a validation methodology for microservice systems. This methodology consists of microservice unit validation and integration validations of microservice systems.

Study [H] focuses on the problem of testing the resilience of MSA-based applications. They present Gremlin, which is a framework for assessing the failure-handling capabilities of microservices. This framework is based on the idea that is about manipulating interservice messages at the network layer while designing and executing tests. This framework has been validated by multiple case studies and their results show that this framework helps uncover bugs in failure recovery code and is suitable for MSA-based systems.

Study [G] addresses the problem of performance tests by checking the needs of non-functional requirements such as response time and throughput. In this paper, the authors propose a new framework that makes performance tests run automatically. This solution is hooked on the HyperText Transfer Protocol (HTTP) and can be built comfortably. It consists of two main aspects, which are a methodology allowing external applications to access the test parameters and a mechanism for using the methodology. The main feature of this proposal is to place the test methodology on each service. According to test results, the mean of the average response time is decreasing compared to the one without the framework.

Study [AL] argues the capabilities of testing frameworks to ensure reliability and quality of applications. In order to overcome the lack of capabilities of the testing framework, the authors decided to provide the automation testing of the microservice with the help of an integrated structure that includes the adaptation layer, data layer, test case layer, execution layer, analysis layer, and management layer. As a result of this study, an automation testing framework is proposed to ensure reliability and quality, and ease of test data generation.

To sum up, testing MSA is an essential step to deliver an application because there are a lot of points that need to be tested. Due to having a distributed nature, it can be complex and difficult to manage. In order to overcome these challenges, an automation testing solution and comprehensive testing framework are proposed. Furthermore, testing failure handling capabilities and microservice unit validation, and functional integration validation approaches are proposed to develop a resilient and reliable application.

5.2.4. Performance Prediction, Measurement, and Optimization

Study [AA] focuses on the dynamic workloads problem and addresses it, and the authors propose an adaptive performance simulation approach. They measure the performance of applications with a queue-based model and then estimate the response time by modifying the parameters of the performance model of the application. To validate this approach, microservice-based applications and simulated workloads are set up. It has been observed according to the experimental results that this approach to performance simulation gives better results in terms of response time compared to other existing methods.

Study [AK] investigates the factors to decrease performance overhead for microservices. Therefore, the authors suggest a three-layer performance model and prediction method and it is built upon performance optimization and modeling. They carried out both experimental and simulation tests to validate the performance model. It is evaluated that this method provides significant advantages to enhance the performance of microservices.

Study [AR] points out the challenge of predicting the workload capacity of microservices. The authors suggest a performance prediction model to address this challenge, and a tool called Terminus was prepared to estimate the capacity of each microservice with respect to different deployments. To evaluate this model, an experiment environment is set up, which consists of four microservices. Experiment results show that it gives good results to predict capacity with a mean absolute percentage error (MAPE) of less than 10%.

Study [K] indicates that the size of a microservice directly impacts its performance and availability. This paper proposes an approach providing workload-based feature clustering for deployment to improve the performance of an MSA. This approach uses a genetic algorithm for clustering. To leverage this approach, they have created microservice architecture deployment optimizer (MicADO), an open-source tool, and this approach has been applied in a case study on an enterprise resource planning (ERP) system. The case study results show that there is a meaningful improvement in the performance of the system.

Study [E] focuses on scalability, manageability, and performance issues. This paper points out that these issues have become more remarkable with the MSA getting popular.

This paper proposes a performance analytical model for a what-if analysis and capacity planning. Finally, two experiments have been conducted to validate this approach by using performance metrics such as response time and probability of request rejection. It has been seen that a what-if analysis and capacity planning for MSA could be applied for minimum cost and time.

Study [BB] offers an approach for assessing scalability and performance on different microservice deployment configurations quantitatively. Additionally, a domain-based metric for each alternative is defined and can be used for making a decision on which one is well-suited. This approach has been evaluated by extensive experiments. The authors note that the domain-based metric for one of the environments is a function that does not increase the number of CPU resources. In addition, they strongly recommend that it is necessary to have and execute performance engineering activities to modify by adding resources to deployment configuration in auto-scaling cloud environments.

Study [BY] points out that there is a lack of predicting performance degradation and its root cause. Although some approaches aim to predict performance degradation, they do not address its root cause. This paper proposes a framework to detect its root cause as well. This framework called SuanMing can predict root causes for potential performance degradation. Further, its aim is to prevent performance degradation before it occurs. To validate this approach, the authors evaluated their framework in two MSA-based systems. Evaluation results confirmed its accuracy of over 90% in predicting performance degradation.

Study [CF] indicated that the number of studies addressing performance problems of MSA-based systems is limited. In this study, the authors propose a model-driven approach for continuous performance enhancements by defining some dedicated metamodels. This study provides refactoring actions that enable performance improvements by taking advantage of the relationship between the monitored data and the architectural model. This approach has been used on two MSA-based systems to validate its feasibility.

In summary, the challenges of this topic are related to performance prediction, measurement, and optimization. With the wide use of microservice applications, it has become possible to create a wider solution set for performance problems. There are studies that address performance issues directly, as well as studies that address other problems such as scalability and load balancing and provide benefits at the point of performance. These studies have been evaluated in their own categories.

5.2.5. Communication and Integration

Study [AI] focuses on complex microservice data flows and communication. To contribute to the solution of this problem, an event-driven lightweight platform called Beethoven for microservice orchestration is proposed. The platform is formed of reference architecture and an orchestration language. To prove its practicality, an example application has been implemented.

Study [R] focuses on the network pressure increase because of inter-microservice communication. The networking of containerized microservice is inefficient. This paper proposes a high-performance user–space networking solution for containerized microservices called DockNet and provides a master–slave threading model to decouple execution and management. This model uses Data Plane Development Kit (DPDK) and customized lightweight IP (LwIP) as the high-performance data plane and TCP/IP stack. Thus, in order to improve network performance, a robust and fast channel between microservices is built. Various experiments are conducted to validate it and as a result of these experiments, DockNet delivers over 4.2×, 4.3×, 5.5× higher performance compared to existing networking solutions.

To sum up, with the widespread use of MSA, the need for communication and integration between microservices has become a challenging point. However, there are not enough studies proposing solutions. We identified only two studies figuring out some solutions to the challenging part of communication. These studies come up with solutions to complex microservice data flows and network performance.

5.2.6. Service Orchestration

Study [L] focuses on the auto-scalability issue and provides a solution called Elascale for managing resources according to workload and application states. However, there is a need for collecting and analyzing performance metrics to manage the scalability of the system. For this purpose, Elasticsearch is used. In this paper, the authors offer architecture and the initial implementation of Elascale. Elascale consists of auto-scalability and monitoring-as-a-service components. Thanks to the monitoring-as-a-service feature, the application stack is monitored and if necessary, the scale in or out process is applied. Additionally, Elascale is an extendable solution so if desired, a new scaling algorithm can be added.

Study [F] addresses the complexity problem of microservices communication and scalability issues. This study proposes to place the business logic in the database to reduce complexity and obtain more scalable services. Its goal is to combine services with data. The proposed model has been validated by conducting a proof of concept study and experimental results show that there is an increase in terms of performance.

Study [AV] investigates the task scheduling and auto-scaling challenges in clouds. The authors noted that existing algorithms are not compatible with a two-layer structure consisting of virtual machines and containers. Therefore, the authors recommend an Elastic Scheduling for Microservices (ESMS) approach with a workflow scheduling algorithm and a statistics-based strategy to find out the best-suited configuration under a continuous workload. To validate this approach, many simulation base experiments have been conducted. Experiment results show that ESMS reduces the cost.

The study [BF] argues how to provide auto-scalability efficiently to reduce costs and energy usage and the authors stated that a good solution will bring a significant increase in performance. Hence, they aim to build an autoscaling system using past service experiences. To this end, they focus on which microservice needs to be scaled for performance improvements. Finally, they propose a pipeline for auto-scaling and also an evaluation of a hybrid sequence and a supervised learning model. According to the experimental result, using a supervised model is so useful for microservices should be scaled up more.

Study [AN] addresses the container resource scheduling challenge. The authors indicated that handling the container resource scheduling problem effectively will decrease the cost and increase the cluster performance. Hence, a multi-objective optimization model with a novel ant colony algorithm for the container-based microservice scheduling is proposed. They aim to improve the metrics related to computing and storage by the proposed ant colony algorithm. To validate this approach, an experiment was conducted and its result shows that the proposed ant colony algorithm for optimization gives good results in terms of load balancing and cluster service reliability.

Study [BA] focuses on utilizing the computing resources challenge. To address this challenge, a container-aware application scheduling strategy is proposed in this paper. The proposed strategy has multiple capabilities composed of using appropriate lightweight containers with minimum deployment cost and a heuristic-based auto-scaling policy for optimizing computing resources. According to evaluation results, the proposed method shows significant improvement compared to existing studies in terms of processing cost, processing time, and resource utilization.

Study [AX] indicates that autoscaling is an important mechanism to manage workload. Computing resource is a key concept for autoscaling because when the workload increases in the system, it should be used in an effective way not to decrease performance. To this end, in this paper, a novel system named Microscaler is proposed to automatically identify the services that need scaling by collecting and analyzing metrics in the application stack and scaling them to manage workload properly. The experimental results in a microservice benchmark show that Microscaler obtains a better result than state-of-the-art methods in terms of optimum service scale and achieves an average of 93% accuracy in determining the service needed in scaling.

Study [AU] analyzes how an orchestration mechanism is integrated into microservice-based cloud applications without making much reengineering. This paper suggests a generic architecture and initial implementation called MICADO to support service orchestration. Additionally, an implementation of this architecture is provided to show its usage and how the scalability of the data avenue file transfer application can be improved. The authors claim that scaling up and down application clusters is made with MICADO effectively.

Study [W] addresses the issues of failure–repair behavior of the containers. In this study, the authors propose different configuration models inspired by Google Kubernetes to deploy software as a container. To make container availability analysis, non-state-space and state-space analytic models are developed. These configuration models are defined by a fail-response and migration service. Additionally, in this paper, an open-source tool is developed by using these models. It helps system administrators to monitor and evaluate containerized system availability.

Study [Y] focuses on the need for supporting the stateful application workloads by providing persistence storage. The authors propose the cloud native storage service orchestration platform based on the IBM ubiquity framework. This platform provides solutions for persistent storage among different container orchestrators and supports multi-protocol access for volume management within the storage systems The authors give the results of the effectiveness of the cloud native storage service orchestration platform by preparing a prototype implementation. They estimate that the proposed framework will be useful for MSA-based systems.

Study [AM] addresses the challenge of heterogeneity of functional and non-functional requirements of microservices. It is important to satisfy all these requirements to overcome these challenges. The authors aim to support the deployment of microservices based on the monitoring. To this end, an architecture is proposed to manage the deployment involving several cloud providers and to find the best deployment plan. Evaluation results show that this approach provides a solution within the expected time interval.

Study [AT] identifies that there is a lack of deployment across data centers because most of the study has worked on the deployment to clusters in data centers. Therefore, a more dynamic approach is necessary to handle the deployment of applications in the edge computing paradigm. To this end, the authors recommend a fully distributed and decentralized orchestrator for containerized microservices, which is called DOCMA. To validate this approach, an experiment was conducted and its result shows that DOCMA has the required ability for the orchestration of microservices.

Study [AE] focuses on the problem of facing unpredictable workloads. The microservice-based application must match as closely as possible to the request to respond quickly and keep costs to a minimum. This paper proposes a novel heuristic adaptation process including two mechanisms that complement each other. While the first mechanism balances the load intensity by scaling containers according to the capability of the process, the latter manages additional containers to handle unpredictable workload changes. The experiment results show that this method manages unpredictable workloads successfully.

Study [N] addresses the load balancing issues and proposes a simple algorithm for a decentralized load balancing system for microservices inside a container used to implement a task executing in a cloud. It can provide better performance compared to the existing centralized container orchestration systems.

Study [Q] addresses the problem of overload control for large-scale microservice-based applications. The authors propose an overload control scheme designed for MSA, called DAGOR. It monitors the load status of each microservice in real-time and distributes the load between the related services when overload is detected. DAGOR has been used in the messaging application for five years. According to experience and experiment results, DAGOR achieved high success.

Study [AW] focuses on the latency because of long-chain microservices. Generally, a request is processed by many microservices called chains, and these microservice chains are in a competition to use resources. The authors noticed that there is not enough study to handle the competition between microservices chains. In this study, a queue-based and chain-oriented load balancing method is proposed. With this method, it is claimed that this method decreases the latency of the long chain. Their evaluations also show that it could decrease the latency of long chains.

Study [BE] addresses the load balancing issue. The authors stated that load balancing is the most important mechanism for availability and scalability and there are some techniques such as client-side and server-side to implement load balancing in a system. However, in order to benefit from each method's advantages, they consider combining them. To this end, they propose a hybrid model to leverage the advantages of both sides.

Study [O] points out the scheduling problem of microservices, especially in multiple clouds. The authors believe that there is an alternative way showing decreased overall turnaround time in contrast to the standard biased greedy scheduling algorithm. For this purpose, they propose an affinity-based scheduling approach and compare it with the standard biased greedy algorithm. The proposed approach achieves a big improvement.

Study [AY] addresses total network delay and network price issues. In addition, the authors noted that increasing energy efficiency is an important task in an edge platform. They aim to minimize network delay and price and improve energy efficiency by designing a novel approach. To this end, they propose a dynamic microservice scheduling algorithm for mobile edge computing (MEC) and evaluate the computational complexity of the scheduling algorithm. According to simulation results, it has been observed that the microservice scheduling framework improves the performance metrics based on total network delay, energy consumption rate (ECR), failure rate, average price, and satisfaction level.

Study [AO] points to increase energy consumption and low service performance with MSA. The authors stated that since resource allocation should be handled efficiently, unlike the current studies not focusing on optimization issues for such chain-oriented service provisioning, they focus on the resource allocation optimization problem. They aim to optimize end-to-end response time and resource usage. To this end, the three-stage scheme is proposed to improve the metrics mentioned above. According to the evaluation, their approach provides a better result than benchmarking algorithms on load balancing and energy consumption.

Study [AQ] works on a novel scheduling framework for Kubernetes. The authors aim to introduce a solution providing improvement for locally main tasks. For this purpose, they propose a hybrid-state scheduler for unscheduled jobs. To validate this approach, they carried out an analysis of their approach's capabilities and evaluation results show that it will overcome problems of their existing solution in their clusters such as collocation interference, priority preemption, high-availability, and baseline scheduling problems.

Study [BD] handles scheduling issues in terms of some concerns such as availability, reliability, resource utilization, scalability, and power consumption. It is noted that current scheduler solutions do not cover all of these concerns. However, the authors claimed that these concerns should be handled together in a scheduling approach to take better results. To this end, they propose a many-objective genetic algorithm scheduler (MOGAS) to handle all these concerns. According to comparison results with ant colony optimization (ACO)-based scheduler, it gives better results in distributing tasks equally and reducing power consumption.

Study [BH] focuses on the problem of determining the accurate resource consumption thresholds to scale applications properly and to ensure high availability. It is also stated that lower thresholds could cause many problems where the services become unavailable against the load. For this purpose, the authors propose a model for calculating the total resource consumption of containers by using mathematical formulas based on Gaussian functions and they managed to calculate the upper threshold values. They use a research project to validate the calculated value being the minimum number of containers to deal with the load.

Study [BI] addresses a challenge of auto-scaling MSA or IoT-based systems. It is also stated that in order to enhance our system availability and reduce cost and energy usage, auto-scaling should be handled in an effective and efficient way. Hence, the authors aim to design a prototype auto-scaling system for MSA-based web applications. As a part of their study, they have developed a pipeline to be used to auto-scale microservices by experimenting with a hybrid sequence and supervised learning model to validate and endorse scaling solutions.

Study [BK] focuses on the difficulty of orchestrating microservices when business processes expand across multiple microservices. Therefore, this study proposes using declarative business processes to coordinate and orchestrate microservices from a data flow perspective. To validate their recommendations, they used the Beethoven platform introduced in study [AI] and demonstrated the usability of this environment for microservice orchestration along with their proposed method.

Study [BM] provides an approach for container-level scalability. Since most cloud applications tend to be containerized every day and are expected to provide near real-time responses, especially in real-time applications, scalability is becoming a real challenge for this kind of application. The threshold values for autoscaling are getting important to ensure scalability efficiently. Kubernetes suggests some techniques for setting thresholds but setting the right values is still a big challenge. To this end, the authors introduce an intelligent autoscaling system including two modules. The first is in charge of identifying resource demands through a generic autoscaling algorithm and the second one is responsible for identifying the autoscaling threshold values by using reinforcement learning agents. To validate their results, they conducted an experiment, and the experiment results show its efficiency compared to the default autoscaling paradigm. Up to 20% enhancements in response time have been measured.

Study [BP] addresses the challenge of selecting the right communication and collaboration pattern for microservices. There are two well-known patterns in the literature right now which are choreography and orchestration. To address this challenge, the authors propose a decision framework to help solution architects to consider key factors and goals. Further, they provide a weighted scoring method to select the most convenient pattern. The requirements of three case studies (Danske Bank, LGB Bank, and Netflix) were reviewed and evaluated to demonstrate this framework's usability. According to the results of their evaluation, a hybrid approach using both patterns has been suggested.

Study [BW] focuses on the challenges of scheduling and autoscaling. The authors claimed existing algorithms had some trouble with streaming workloads and the two-layer structures consisting of virtual machines and containers. Therefore, they propose an elastic scheduling algorithm to overcome these challenges. This algorithm handles task scheduling and auto-scaling, which is based on a variable-sized bin packing problem (VSBPP). With the conducted experiments, the proposed algorithm has been validated that proposed algorithm improves the success ratio and cost.

Study [BX] points out that remote registry-based images could cause increased pulling traffics and startup time latency. To solve these issues, the authors come up with an idea of layer sharing deployment for microservices. Since containers are generally implemented as multi-layered structures, they claim that common layers can be shared between microservices. For this purpose, they propose an accelerated distributed augmented lagrangian (ADAL) based algorithm to be used by servers and registries. Experiment results show that it reduces the microservice startup time by 2.20 times on average.

Study [BZ] addresses the issue of performance degradation when traffic increases. The authors claim that existing approaches of autoscaling do not pay enough attention to the microservice chain and performance degradation issues. This study proposes an autoscaling framework for microservice chains. It includes two modules. The first is responsible for collecting samples from microservices and training a latency model using the GNN. The second is responsible for identifying the number of microservice instances through the GNN model. Their evaluation results demonstrate pHPA effectiveness with reduced latency and improved resource usage.

Study [CA] notes that the flow control rules are generally adjusted and applied manually. In addition, it is also noted that availability is really critical for MSA-based systems and it should be handled with some concepts such as fault tolerance and flow limiting. To improve the availability of a system, the authors claim that flow control rules should be handled dynamically. To this end, they propose a dynamic flow control algorithm. The algorithm works on monitoring data and current flow and determines the flow-limiting thresholds. Evaluation results show that automatic flow control mechanisms obtain better results in terms of performance compared to traditional static methods.

Study [CB] proposes a microservice rescheduling framework to address performance degradation and response time challenges. The authors point out that response time is one of the most important keys to quality of service. Hence, runtime adaptations and rescheduling should be handled carefully. They stated that existing works lack handling the effect of configuration parameters of container-based microservices. The proposed solution makes some periodic monitoring and then rescheduling activities are triggered based on threshold-based rules. Experiment results demonstrate that with the proposed framework, a significant reduction of up to 13.97% in the average response rate was achieved.

Study [CD] focuses on the availability issues of MSA-based systems. It points out that availability is still a problem while migrating a legacy application to MSA even if microservices will be running on Kubernetes, which is a popular service orchestration platform. The authors stated that repair actions of Kubernetes cannot satisfy the high availability (HA) requirements. Hence, they propose an approach in which automatic service redirection to healthy microservices and application state replication can be achieved by adding service recovery to the repair actions of Kubernetes. Their experimental results show that their solution brings an improvement in terms of response time.

To sum up, there are many kinds of studies addressing almost all the concerns about deployment, scheduling, auto-scalability, load-balancing, and orchestration. These are the most important areas in service orchestration. Additionally, these studies use some best practices and technologies implemented by some big vendors. Thus, it allows these studies to be used in a wide application area.

5.2.7. Security

Study [AJ] focuses on the limitations of access control technologies in the microservice environment. This paper suggests an access control optimization model based on role-based access control (RBAC). This model enhances the attribute-based encryption (ABE) model being one of the most common cryptographic mechanisms, in which existing RBAC users can directly access the ABE encrypted data in microservices. It has several advantages compared to ABE, which includes improving the expression ability of access policies, the security and operational efficiency of microservices, and reducing the computational cost.

Study [V] investigates the microservices security topic and tries to identify the taxonomy of security issues. While making this research, Docker Swarm and Netflix security decisions are also investigated. This paper claims that microservice security requires a layered security solution consisting of hardware, virtualization, cloud, communication, application, and orchestration. In this paper, a prototype framework for microservice security is described and a case study is conducted. The case study result shows the performance overhead of the security is around 11%.

Study [D] focuses on two problems of microservice security. First, network complexity complicates monitoring the security. Second, due to the trust among microservices, if any microservice fails, it may affect the entire application. In this paper, the authors propose a design for security-as-a-service for microservices-based cloud applications and they implement a flexible monitoring and policy enforcement infrastructure for network traffic by adding a new application programming interface (API) primitive FlowTap for the network hypervisor. Effectiveness analysis results show that the proposed solution is able to tackle various monitoring scenarios.

Study [AC] addresses the problems of authentication and authorization in the 5G platform. The authors come up with a solution based on specifically identity and access control of microservices. The proposed solution has been implemented in the network function virtualization (NFV) based platform called SONATA. It encourages using well-known techniques and simple designs for identity and access control and favors role-based access control.

Study [T] focuses on the problems of the authenticity and confidentiality of microservice calls. This paper criticizes the HTTP-based approach used for microservice and API calls and transport layer security (TLS), providing only link-level channel security. In order to prevent these security problems, this paper comes up with a solution consisting of authentication with password and key pair and decentralized role-based authorization.

Study [BT] points out that the security of access control becomes challenging as the system grows because it causes more access points to be handled for security. The authors propose an extended version of role-based access control (RBAC) called hierarchical trust RBAC. This model enables security managers to detect unauthorized access to sensitive information and identify verification. They also conducted a case study to show the feasibility of their model. Case study results showed that it provides faster and more flexible access to sensitive information.

In summary, secure microservices are tightly dependent on our MSA design. Software architects should design MSA by taking into account the security concern since it might be tough work to provide a secure system later. These studies propose solutions to identified problems, but there is no study proposing a model about how to design MSA to ensure security.

5.2.8. Monitoring, Tracing, and Logging

Study [X] focuses on managing complex dependency relationships between microservices. This paper proposes an approach called graph-based microservice analysis and testing (GMAT) that automatically prepares a service dependency graph (SDG). It allows us to analyze, visualize and trace the dependency relationships between microservices. Additionally, it allows detecting anomalies by watching service invocation chains. Experiment results show that GMAT is capable of managing complex dependency relationships for MSA-based systems.

Study [Z] addresses the problem of complex interactions, identifying abnormal services. Hence, the authors present a novel system called Microscope to efficiently generate a service causal graph and extract the causes of performance problems. Experimental evaluations show that Microscope has a good result and it is also claimed that it is better than most recent technology solutions.

Study [AD] addresses that extracting component relations from just static sources is not enough for the accurate result because component relationships might arise at runtime. Extracting component relations is important to detect design drawbacks or potential architectural improvements. In order to overcome these issues, the authors offer an approach to extract and analyze the architecture of an MSA-based software system according to not only static service information but also aggregated runtime information. They have conducted an experiment to evaluate the approach. The results show that this approach is useful for detecting design drawbacks and improving the design.

Study [AS] focuses on the problem of heterogeneity of logs. In other words, each microservice can create logs in a different format and it causes heterogeneity for logs. It is tough work to understand and interpret these logs to make the right decision for the system. Therefore, this paper suggests a novel approach based on representational state transfer (REST) architecture style. Two case studies have been made to validate an approach and evaluate an implementation of this approach called MetroFunnel. The assessment results indicated that it is successful in traversing logs and reducing the size of collected data.

Study [P] addresses the importance of high decoupling among microservices because the authors realized that there is a lack of highlighting microservice communications. Hence, an architecture recovery tool called MicroART is presented in this study to show communications among microservices. This tool consists of four main components which are Docker Analyzer, GitHub Analyzer, Log Analyzer, and Model Log Analyzer to be able to generate the models. The authors indicated that it can be used by software architects for analysis, documentation, and architectural reasoning.

Study [AZ] investigates the root cause of anomalies in the application and it is stated that it can be a complicated and time-consuming job because a lot of communications need to be investigated. In this work, the root cause analysis framework is recommended which is graph-based. In order to show the effectiveness of this framework Grid'5000 testbed has been used to deploy three different architectures and then some anomalies were injected into these architectures. The evaluation result shows that this approach is more effective than a machine learning method ignoring the relationship between elements.

Study [BG] also focuses on anomaly detection and its root cause in MSA. Authors claim that most studies regarding root cause analysis (RCS) mainly address data monitoring, data dependency among services, and invocation data. However, in this study, they use invocation chain anomaly analysis to address the RCA problem. They have implemented an algorithm based on robust principal component analysis and a single indicator anomaly detection algorithm. They run their algorithm on a batch of sample data and three batches of test data of the 2020 International AIOps Challenge. They got a good score according to the scoring criteria of organizers and their algorithm put in a good performance with higher accuracy than some other traditional algorithms used in this area for anomaly detection.

Study [BJ] points out the challenges experienced in traceability analysis in MSA base systems. Since it is a complex and dynamic environment, analyzing to investigate any problem can be challenging. This is mostly due to the fact that there is too much trace data and it is difficult to obtain the necessary information to detect the real problem. Therefore, the authors recommend a graph-based approach for trace analysis. The strength of their proposed method is that it provides efficient processing and storage, as well as a powerful access mechanism by combining graph database and real-time analytics database. They have conducted an experiment to validate their approach and the results of the experiment have confirmed its efficiency and effectiveness in diagnosing the problem.

Study [BL] focuses on the system-wide challenges of observability. The distributed and heterogeneous nature and tendency to decentralize responsibility are the factors that complicate the observability of MSA-based systems. In this study, the challenges of providing observability in MSA-based systems are emphasized and an offline approach that performs distributed tracing is proposed. With this method, it is recommended to model microservices as observable execution paths, so abstraction is provided to generate realistic trace data again.

Study [BN] proposes a novel layered diagnosis framework including a service response layer, timing constraints, causality analysis, and a ranking algorithm for detecting faulty microservices. The authors indicated that as system size grows, detecting faulty microservices in a complex environment would get challenging. Thus, they claim that their framework could be a solution to this problem. They also carried out a case study to validate their approach. Experimental results show that it managed to achieve 89% specificity and 77% recall.

Study [BO] provides a monitoring solution called Kmon for MSA-based systems. The authors aim to monitor the complex microservice environment and internal states of microservices in an effective way with their proposed solution. This solution collects indicators by breaking them into three categories: TCP request data, topology level, and the other indicators related to CPU, memory, block I/O, etc. To validate the proposed solution, the authors conducted an experiment. Experiment results show that it has little effect on response time and low CPU usage.

Study [BQ] points out the challenge of detecting faulty microservices and root cause localization. For this purpose, the authors propose a method called Microservice Fault Root Cause Location Method Based on Correlation Analysis (MFRL-CA). In this method, a microservice fault propagation graph is built by collecting the correlation between historical fault data and dependent call data to reduce the time consumption of detecting faulty services. They carried out an experiment to show their approach's effectiveness and the results show that this method effectively managed to detect faulty services and their root cause.

Study [BR] proposes a root cause localization framework called ModelCoder. In this study, the authors have introduced some concepts to figure out the root cause of localization problems and developed the framework for these concepts. The first one is a concept for building dependency graphs between microservices. The second one is a formulization for root cause localization problems based on the graph built in the first step. Finally, a fault model called ModelCoder is built on these two concepts. They evaluated ModelCoder on a real-world system and the results show that ModelCoder is able to detect faulty root nodes within 80 s on average.

Study [BS] also points out the root cause of localization problems and aims to detect microservice failures in an effective way. For this purpose, the authors come up with a method for detecting microservice failures by using a semi-supervised learning model and dynamic sliding window methods. To evaluate their model, they used public data and the results showed that the model had a good performance and the accuracy of anomaly detection and root cause location was close to 100%.

Study [BU] addresses availability issues caused by service anomalies. The authors stated that existing approaches were limited in terms of inefficient traversing mechanism of service dependency graph and detecting anomalies process also could result in failure. To this end, they propose a highly efficient root cause localization approach based on dynamically constructed service call graphs. Experimental results and the result of being used in Alibaba showed it obtained good results in terms of accuracy and efficiency.

Study [BV] aims to address the root cause of performance issues. The authors claim that complex communication among services makes the system performance unpredictable and hard to trace and detect the root cause of performance issues. Therefore, they propose a tool called T-Rank. It uses tracking data and combines them with a tracing chain. Further, it provides a ranked suspicious list of the containers based on the spectrum algorithm. As a result of their experiment with the data collected from a real-world MSA-based system, T-Rank is feasible to be used in MSA base system thanks to its high accuracy and low resource cost.

The authors [CE] propose an anomaly detection approach for MSA-based systems. They stated that existing approaches do not have the required skills to detect faulty services accurately. Therefore, they propose an anomaly detection approach. In this approach, first, execution traces are collected across microservices, then the anomaly degree of traces is calculated and then differences between traces are analyzed to locate the components causing anomalies. According to their evaluation results, this approach achieves high precision and recall in detecting anomalies.

Study [CG] provides an agent-based monitoring platform by monitoring not only internally developed services but also externally developed services with the help of sidecar containers. Agents are responsible for monitoring incoming and outgoing network traffic and also system state by reading kernel data. Prototype evaluation results show that their solution has a similar performance as Prometheus, but also, they offer some functionalities focused on multi-vendor service integration.

In summary, it is important to monitor the environment after developing microservices, so these studies in this part generally focus on the monitoring architecture model, extract dependencies, and anomaly detection. There is a lack of a powerful tool with integration API with other third-party software among these studies.

5.2.9. Decomposition

Study [AB] focuses on deciding the right size of microservices and provides a conceptual methodology to decompose business capability based on domain-driven design principles. To evaluate the usage of this methodology, a case study is conducted on the weather information dissemination domain. Evaluation results show that the weather information dissemination system is partitioned into different microservices successfully.

Study [U] proposes a systematic approach using functional decomposition and based on functional requirements. This approach aims to build high cohesive and low coupled decomposition. To evaluate this approach, they have compared microservices implementations by three independent teams. The evaluation results show that it achieves identifying microservices much faster.

The authors [AP] stated that the decomposition process is so challenging task and it should be supported with an approach, so they suggest a data flow-driven decomposition approach to handle the decomposition problem of MSA. They aim to obtain independently deployable and scalable microservices, so they defined a four-step decomposition procedure consisting of business requirement analysis, building fine-grained data flow diagrams, extracting dependencies between processes and finally identifying microservices by clustering processes. They conducted a case study to validate this approach and it has been observed that microservice candidates are determined by taking coupling and cohesive constraints into consideration.

Study [CC] proposes an approach for identifying microservices by analyzing dependencies between business processes thanks to control, data, and semantic models. Further, it also provides a clustering method to identify potential microservices. To validate this approach, the authors carried out a case study. The results of the case study demonstrate its doability. Additionally, it also achieves better results than existing approaches in terms of microservice identification.

In summary, the better we decompose the business capability into microservices, the more powerful microservices we have so we can say that this challenge is the primary among other challenges. Despite this fact, we could not find enough studies to work on this topic deeply.

6. Discussion

We conducted a systematic literature review following the guidelines of Kitchenham et al. [15]. The main purpose of the SLR was to identify relevant challenges and solution directions. For this purpose, we conducted a comprehensive study and selected 85 primary studies from 3842 papers. We have carefully applied selection and elimination criteria in order to catch the most relevant studies for our SLR study. As a result of our study, we could explore nine problem categories. We have observed that each study has addressed one or more problems and explained the solution to problems in their study. An important number of these problems are related to quality attributes such as reliability, availability, scalability, and performance. We have reported quality concerns related to identified challenges. This SLR could be input to further studies that highlight the relevance of the quality concerns in MSA. It could also show direction to identify which quality concerns have not yet been explicitly addressed. Nevertheless, the fact that no in-depth research has been carried out on these quality concerns does not necessarily imply that they are not relevant for MSA. Therefore, this observation could typically initiate further research on the quality concerns in MSA.

We have observed that with the usage of cloud computing, the cost of resources has emerged as an important topic, so optimization of resource usage and performance and scheduling problems have become crucial. In addition, it has been observed that the challenges of service orchestration and monitoring have been covered in many more studies in recent years and detailed and comprehensive solutions have been presented in those areas. We see this as an expected consequence of any system development process. Since as the systems get bigger and more complex and the need for scalability increases, the need for monitoring starts to occur in those systems, and in parallel, the orchestration needs to increase. This is also the case in the development process of MSA-based systems. We consider these challenges as newly recognized challenges as a result of the growth and complexity of MSA-based systems.

The main threats to the validity [18] of this SLR are related to publication and selection bias, and also to data extraction and synthesis. The publication bias is about the likelihood of the researchers to publish positive results rather than negative ones, which is beyond our control and remains an open issue for future work. We carefully identified and applied the inclusion/exclusion criteria during the screening and review of the primary studies. However, the subjectivity in defining the criteria and selecting the primary studies could have introduced a threat to the validity of this study. To reduce the bias regarding the inclusion/exclusion criteria, we first picked a random set of 10 studies as suggested by [19] and defined the selection criteria. The evaluation and the selection of the primary studies were performed by the first author and selectively reviewed by the co-authors in a randomized manner. Any difference in the selection of the primary studies was discussed in detail and a final decision was reached per study. After the primary studies were evaluated and selected, the relevant data for a pilot set of primary studies were extracted by the first author using a data extraction sheet and taking informative notes on it. The pilot data extraction was then reviewed by the co-authors and conflicts were resolved again by discussions among the authors until a common understanding was reached. Regarding the data synthesis, we applied a systematic grouping of the extracted data on the sheet. The problem categories and their rationale were reviewed and discussed by the authors in meetings, therefore the categories that we identified can be considered to cover the main problems. However, some problems could be considered sub-categories of the basic categories. To highlight these, we have adopted the feature models.

Our goal with this study was the synthesis of the results from the primary studies selected in the SLR. Hence, an in-depth comparison of approaches for each area has been omitted. The findings as such are thus the reported results from the primary studies. A further study and synthesis of these results would be interesting and lead to new primary studies. We consider this as part of our future work.

In this study, we did not elaborate in detail on the implementation platforms for microservices. In our earlier study [20], we have provided an approach for developing platform selection rules for model-driven architecture. The same approach could be applied to microservice platforms. We consider this as part of our future work. Related to MDA, in our recent work [21] we have proposed a model-driven architecture approach for the automated deployment of microservices. The automation approach could be considered a solution for several of the identified challenges. In [22] we have provided a feature-driven characterization of microservice architectures and used this framework to provide an overview of the state of the practice.

7. Related Work

Pahl and Jamshidi [23] aimed to identify, taxonomically classify and compare the existing research and application of microservices. For this purpose, they conducted a systematic mapping study of 21 primary studies. They used a classification framework to sort out the research and extract the keywords according to specified groups.

Soldani et al. [24] presented systematic grey literature on the pains and gains of a microservices topic. They realized that academic work on his topic was at an early stage. They reviewed 51 sources of grey literature from three different perspectives based on three research questions. Next, the authors presented two taxonomy plans for pains and gains of microservices to classify the studies. Finally, they presented the coverage of pains and gains in previous surveys on microservices.

Alshuqayran et al. [25] presented a systematic mapping study of microservice architectures and their implementation. Overall, 33 primary studies were collected for detailed review. Two qualitative and quantitative synthesis methods were used and three research questions were defined to explore the challenges of MSA, which are used to model MSA and its possible quality attributes.

Di Francesco et al. [6] aimed to provide a survey investigating relationships among research contributions on microservices. They performed a systematic mapping of 103 primary studies and produced a clear overview of the state of the art in architecting with microservices. They used three main perspectives to investigate the research on MSA. These were publication trends, the focus of research, and the potential for industrial adoption. Additionally, the authors performed a detailed trend analysis of the data to understand how architecting with microservices has been evolving over time.

Vural et al. [26] performed a systematic literature review to determine the main practical motivations and emerging standards behind the MSA. They performed a systematic literature review of 37 primary studies. As a result of their study, they proposed that there would be an increasing trend soon and there were not enough empirical studies to clarify some challenging topics. In addition, they reported that there was no study targeting the weak points of MSA in particular.

Bushong et al. [27] conducted a systematic mapping study to observe the architecture evolution of microservices and methods of microservice analysis. They selected 55 articles published from 2018 to 2021. In their study, they provide a categorization of papers by five analytical approaches and seven categories for microservice analysis. Another mapping study was carried out with a focus on MSA in DevOps [28].

In this SLR study, unlike other studies, the challenging points in the microservice architecture are deeply examined by adopted feature models and the solution directions to these challenges are explained. In this regard, it is intended as a valuable guide for both academics and practitioners. Table 7 highlights the addressed research questions of the related secondary studies.

Table 7. Comparison of related works with this SLR study.

Study Name	**# Primary Studies**	**Research Questions**
Microservices: A Systematic Mapping Study [20]	21	What are the main practical motivations behind using microservices? What are the different types of microservice architectures? 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?
The pains and gains of microservices: A systematic grey literature review [21]	51	How much evidence of microservices experimentation from the industry is available online? What are the technical and operational "pains" of microservices? What are the technical and operational "gains" of microservices?
A Systematic Mapping Study in Microservice Architecture [22]	33	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?
Architecting with microservices: A systematic mapping study [6]	103	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?
A Systematic Literature Review on Microservices [23]	37	What type of research is conducted on microservices? What are the main practical motivations behind microservices-related research? What are the emerging standards and de facto tools for microservices solutions?
On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study [24]	55	What methods and techniques are used in microservice analysis? What are the problems or opportunities that are addressed using microservice analysis techniques? Does microservice analysis overlap with other areas of software analysis, or are new methods or paradigms needed? What potential future research directions are open in the area of microservice analysis?
Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review	85 (This study)	What are the identified challenges of microservice architectures? What are the proposed solution directions?

Complementary to this study, various other secondary studies have been published on the key elements of microservice architectures [29,30].

8. Conclusions

In this article, the results from a systematic literature review are provided in order to deeply explain the state of the art of MSA and identify the challenges faced in applying it. We have taken into account the studies published since the introduction of MSA in 2014, identified 85 of the 3842 papers discovered as primary studies, and reviewed them in relation to the research questions.

We can say that the application of MSA is becoming more and more popular and brings many solutions and provides important benefits for service-oriented and cloud applications. The focus of this SLR was mainly on the challenges that are encountered when applying MSA. Accordingly, we have identified nine basic categories of problems that were discussed in the selected primary studies. We have synthesized and explained each problem and related solution directions comprehensively and also adopted diagrams to provide an overview of the identified problems and the suggested solutions.

We believe that the findings and results of this SLR will lead to opportunities for further research in MSA. For example, the challenges can be considered to identify new research questions, and the outputs of the study can be used to improve MSA specifications or support practitioners in their decisions in applying MSA. In addition, identified challenges and solution directions provide an overview of the overall picture that could help to analyze different alternative solutions. Accordingly, in our upcoming work, we plan to synthesize a reference model for MSA by considering the problem and solution categories that we have explained in this review. In addition, similarities and differences among the solutions available for the same challenge are helpful information to pick one solution instead of another solution. Therefore, we also plan to analyze both academic and grey literature solutions to give comparative research from this point of view.

Author Contributions: Conceptualization, M.S., B.T. and A.K.T.; methodology, M.S., B.T. and A.K.T.; software, M.S.; validation, M.S., B.T. and A.K.T.; writing-review and editing, M.S., B.T. and A.K.T.; supervision, B.T. and A.K.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Appendix A. List of Primary Studies

A. D. I. Savchenko, G. I. Radchenko, and O. Taipale, “Micro-services validation: Mjolnirr platform case study,” in 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), May 2015, pp. 235–240, doi:10.1109/MIPRO.2015.7160271.

B. M. Rahman and J. Gao, “A Reusable Automated Acceptance Testing Architecture for Micro-services in Behavior-Driven Development,” in 2015 IEEE Symposium on Service-Oriented System Engineering, Mar. 2015, pp. 321–325, doi:10.1109/SOSE.2015.55.

C. N. Viennot, M. Lécuyer, J. Bell, R. Geambasu, and J. Nieh, “Synapse,” in Proceedings of the Tenth European Conference on Computer Systems—EuroSys ’15, 2015, pp. 1–16, doi:10.1145/2741948.2741975.

D. Y. Sun, S. Nanda, and T. Jaeger, “Security-as-a-Service for Micro-services-Based Cloud Applications,” in 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Nov. 2015, pp. 50–57, doi:10.1109/CloudCom.2015.93.

E. H. Khazaei, C. Barna, N. Beigi-Mohammadi, and M. Litoiu, "Efficiency Analysis of Provisioning Micro-services," in 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Dec. 2016, pp. 261–268, doi:10.1109/CloudCom.2016.0051.

F. A. Messina, R. Rizzo, P. Storniolo, M. Tripiciano, and A. Urso, "The Database-is-the-Service Pattern for Micro-service Architectures," Springer, Cham, 2016, pp. 223–233.

G. A. de Camargo, I. Salvadori, R. dos S. Mello, and F. Siqueira, "An architecture to automate performance tests on micro-services," in Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services—iiWAS '16, 2016, pp. 422–429, doi:10.1145/3011141.3011179.

H. V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, and V. Sekar, "Gremlin: Systematic Resilience Testing of Micro-services," in 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Jun. 2016, pp. 57–66, doi:10.1109/ICDCS.2016.11.

I. K. B. Long, H. Yang, and Y. Kim, "ICN-based service discovery mechanism for micro-service architecture," in 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), Jul. 2017, pp. 773–775, doi:10.1109/ICUFN.2017.7993899.

J. S. Haselböck, R. Weinreich, and G. Buchgeher, "Decision guidance models for micro-services," in Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems—ECBS '17, 2017, pp. 1–10, doi:10.1145/3123779.3123804.

K. S. Klock, J. M. E. M. Van Der Werf, J. P. Guelen, and S. Jansen, "Workload-Based Clustering of Coherent Feature Sets in Micro-service Architectures," in 2017 IEEE International Conference on Software Architecture (ICSA), Apr. 2017, pp. 11–20, doi:10.1109/ICSA.2017.38.

L. H. Khazaei, R. Ravichandiran, B. Park, H. Bannazadeh, A. Tizghadam, and A. Leon-Garcia, "Elascale: autoscaling and monitoring as a service," Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering. IBM Corp., pp. 234–240, 2017, Accessed: Jun. 11, 2019. [Online]. Available: https://dl.acm.org/citation.cfm?id=3172823.

M. N. H. Do, T. Van Do, X. Thi Tran, L. Farkas, and C. Rotter, "A scalable routing mechanism for stateful micro-services," in 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Mar. 2017, pp. 72–78, doi:10.1109/ICIN.2017.7899252.

N. M. Rusek, G. Dwornicki, and A. Orłowski, "A Decentralized System for Load Balancing of Containerized Micro-services in the Cloud," Springer, Cham, 2017, pp. 142–152.

O. D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Multi-objective scheduling of micro-services for optimal service function chains," in 2017 IEEE International Conference on Communications (ICC), May 2017, pp. 1–6, doi:10.1109/ICC.2017.7996729.

P. G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, and A. Di Salle, "MicroART: A software architecture recovery tool for maintaining micro-service-based systems," in Proceedings—2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings, Jun. 2017, pp. 298–302, doi:10.1109/ICSAW.2017.9.

Q. H. Zhou et al., "Overload Control for Scaling WeChat Micro-services," in Proceedings of the ACM Symposium on Cloud Computing—SoCC '18, 2018, pp. 149–161, doi:10.1145/3267809.3267823.

R. X. Luo, F. Ren, and T. Zhang, "High Performance Userspace Networking for Containerized Micro-services," Springer, Cham, 2018, pp. 57–72.

S. X. Limon, A. Guerra-Hernandez, A. J. Sanchez-Garcia, and J. C. Perez Arriaga, "Saga-MAS: A Software Framework for Distributed Transactions in the Micro-service Architecture," in 2018 6th International Conference in Software Engineering Research and Innovation (CONISOFT), Oct. 2018, pp. 50–58, doi:10.1109/CONISOFT.2018.8645853.

T. K. Jander, L. Braubach, and A. Pokahr, "Defense-in-depth and Role Authentication for Micro-service Systems," Procedia Comput. Sci., vol. 130, pp. 456–463, Jan. 2018, doi:10.1016/J.PROCS.2018.04.047.

U. S. Tyszberowicz, R. Heinrich, B. Liu, and Z. Liu, "Identifying Micro-services Using Functional Decomposition," Springer, Cham, 2018, pp. 50–65.
V. T. Yarygina and A. H. Bagge, "Overcoming Security Challenges in Micro-service Architectures," in 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), Mar. 2018, pp. 11–20, doi:10.1109/SOSE.2018.00011.
W. S. Sebastio, R. Ghosh, and T. Mukherjee, "An Availability Analysis Approach for Deployment Configurations of Containers," IEEE Trans. Serv. Comput., pp. 1–1, 2018, doi:10.1109/TSC.2017.2788442.
X. S.-P. Ma, C.-Y. Fan, Y. Chuang, W.-T. Lee, S.-J. Lee, and N.-L. Hsueh, "Using Service Dependency Graph to Analyze and Test Micro-services," in 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Jul. 2018, pp. 81–86, doi:10.1109/COMPSAC.2018.10207.
Y. A. Warke, M. Mohamed, R. Engel, H. Ludwig, W. Sawdon, and L. Liu, "Storage Service Orchestration with Container Elasticity," in 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Oct. 2018, pp. 283–292, doi:10.1109/CIC.2018.00046.
Z. J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments," Springer, Cham, 2018, pp. 3–20.
AA. Y. Sun, L. Meng, P. Liu, Y. Zhang, and H. Chan, "Automatic Performance Simulation for Micro-service Based Applications," Springer, Singapore, 2018, pp. 85–95.
AB. E. Bainomugisha, A. S. I. G. on S. Engineering, and ACM Digital Library., Partitioning Micro-services: A Domain Engineering Approach. ACM, 2018.
AC. D. Guija and M. S. Siddiqui, "Identity and Access Control for micro-services based 5G NFV platforms," in Proceedings of the 13th International Conference on Availability, Reliability and Security—ARES 2018, 2018, pp. 1–10, doi:10.1145/3230833.3233255.
AD. B. Mayer and R. Weinreich, "An Approach to Extract the Architecture of Micro-service-Based Software Systems," in 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), Mar. 2018, pp. 21–30, doi:10.1109/SOSE.2018.00012.
AE. F. Klinaku, M. Frank, and S. Becker, "CAUS: An Elasticity Controller for a Containerized Micro-service" in Companion of the 2018 ACM/SPEC International Conference on Performance Engineering—ICPE '18, 2018, pp. 93–98, doi:10.1145/3185768.3186296.
AF. K. Jander, A. Pokahr, L. Braubach, and J. Kalinowski, "Service Discovery in Megascale Distributed Systems," Springer, Cham, 2018, pp. 273–284.
AG. M. J. Kargar and A. Hanifizade, "Automation of regression test in micro-service architecture," in 2018 4th International Conference on Web Research (ICWR), Apr. 2018, pp. 133–137, doi:10.1109/ICWR.2018.8387249.
AH. G. Pardon, C. Pautasso, and O. Zimmermann, "Consistent Disaster Recovery for Micro-services: the BAC Theorem," IEEE Cloud Comput., vol. 5, no. 1, pp. 49–59, Jan. 2018, doi:10.1109/MCC.2018.011791714.
AI. D. Monteiro, R. Gadelha, P. H. M. Maia, L. S. Rocha, and N. C. Mendonça, "Beethoven: An Event-Driven Lightweight Platform for Micro-service Orchestration," Springer, Cham, 2018, pp. 191–199.
AJ. G. Fu, J. Sun, and J. Zhao, "An optimized control access mechanism based on micro-service architecture," in 2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2), Oct. 2018, pp. 1–5, doi:10.1109/EI2.2018.8582628.
AK. L. Bao, C. Wu, X. Bu, N. Ren, and M. Shen, "Performance Modeling and Workflow Scheduling of Micro-service-based Applications in Clouds," IEEE Trans. Parallel Distrib. Syst., pp. 1–1, 2019, doi:10.1109/TPDS.2019.2901467.
AL. Y. Wang, L. Cheng, and X. Sun, "Design and Research of Micro-service Application Automation Testing Framework," in Proceedings—2019 International Conference on Information Technology and Computer Application, ITCA 2019, Dec. 2019, pp. 257–260, doi:10.1109/ITCA49981.2019.00063.

AM. E. Fadda, P. Plebani, and M. Vitali, "Monitoring-aware Optimal Deployment for Applications based on Micro-services," IEEE Trans. Serv. Comput., pp. 1–1, Jul. 2019, doi:10.1109/tsc.2019.2910069.

AN. M. Lin, J. Xi, W. Bai, and J. Wu, "Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Micro-service Scheduling in Cloud," IEEE Access, vol. 7, pp. 83088–83100, 2019, doi:10.1109/ACCESS.2019.2924414.

AO. Y. Yu, J. Yang, C. Guo, H. Zheng, and J. He, "Joint optimization of service request routing and instance placement in the micro-service system," J. Netw. Comput. Appl., vol. 147, p. 102441, Dec. 2019, doi:10.1016/j.jnca.2019.102441.

AP. S. Li et al., "A dataflow-driven approach to identifying micro-services from monolithic applications," J. Syst. Softw., vol. 157, p. 110380, Nov. 2019, doi:10.1016/j.jss.2019.07.008.

AQ. O.-M. Ungureanu, C. Vlădeanu, and R. Kooij, "Kubernetes cluster optimization using hybrid shared-state scheduling framework," in Proceedings of the 3rd International Conference on Future Networks and Distributed Systems—ICFNDS '19, 2019, Accessed: Aug. 27, 2020. [Online]. Available: https://doi.org/10.1145/3341325.3341992.

AR. A. Jindal, V. Podolskiy, and M. Gerndt, "Performance modeling for cloud micro-service applications," in ICPE 2019—Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, Apr. 2019, pp. 25–32, doi:10.1145/3297663.3310309.

AS. M. Cinque, R. Della Corte, and A. Pecchia, "Micro-services Monitoring with Event Logs and Black Box Execution Tracing," IEEE Trans. Serv. Comput., 2019, doi:10.1109/TSC.2019.2940009.

AT. L. L. Jimenez and O. Schelen, "DOCMA: A decentralized orchestrator for containerized micro-service applications," in Proceedings—2019 3rd IEEE International Conference on Cloud and Fog Computing Technologies and Applications, Cloud Summit 2019, Aug. 2019, pp. 45–51, doi:10.1109/CloudSummit47114.2019.00014.

AU. T. Kiss et al., "MiCADO—Micro-service-based Cloud Application-level Dynamic Orchestrator," Futur. Gener. Comput. Syst., vol. 94, pp. 937–946, May 2019, doi:10.1016/J.FUTURE.2017.09.050.

AV. S. Wang, Z. Ding, and C. Jiang, "Elastic Scheduling for Micro-service Applications in Clouds," IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 1, pp. 98–115, Jul. 2020, doi:10.1109/tpds.2020.3011979.

AW. F. Wan, X. Wu, and Q. Zhang, "Chain-Oriented Load Balancing in Micro-service System," in 2020 World Conference on Computing and Communication Technologies (WCCCT), May 2020, pp. 10–14, doi:10.1109/WCCCT49810.2020.9169996.

AX. G. Yu, P. Chen, and Z. Zheng, "Microscaler: Cost-effective Scaling for Micro-service Applications in the Cloud with an Online Learning Approach," IEEE Trans. Cloud Comput., pp. 1–1, Apr. 2020, doi:10.1109/tcc.2020.2985352.

AY. A. Samanta and J. Tang, "Dyme: Dynamic Micro-service Scheduling in Edge Computing Enabled IoT," IEEE Internet Things J., pp. 1–1, Mar. 2020, doi:10.1109/jiot.2020.2981958.

AZ. Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and micro-service architectures," J. Syst. Softw., vol. 159, p. 110432, Jan. 2020, doi:10.1016/j.jss.2019.110432.

BA. S. N. Srirama, M. Adhikari, and S. Paul, "Application deployment using containers with auto-scaling for micro-services in cloud environment," J. Netw. Comput. Appl., vol. 160, p. 102629, Jun. 2020, doi:10.1016/j.jnca.2020.102629.

BB. A. Avritzer et al., "Scalability Assessment of Micro-service Architecture Deployment Configurations: A Domain-based Approach Leveraging Operational Profiles and Load Tests," J. Syst. Softw., vol. 165, p. 110564, Jul. 2020, doi:10.1016/j.jss.2020.110564.

BC. A. De Iasio and E. Zimeo, "A framework for micro-services synchronization," Softw. Pract. Exp., p. spe.2877, Aug. 2020, doi:10.1002/spe.2877.

BD. M. Imdoukh, I. Ahmad, and M. Alfailakawi, "Optimizing scheduling decisions of container management tool using many-objective genetic algorithm," Concurr. Comput. Pract. Exp., vol. 32, no. 5, Mar. 2020, doi:10.1002/cpe.5536.

BE. M. Autili, A. Perucci, and L. De Lauretis, "A Hybrid Approach to Micro-services Load Balancing," in Micro-services, Springer International Publishing, 2020, pp. 249–269.

BF. N. C. Coulson, S. Sotiriadis, and N. Bessis, "Adaptive Micro-service Scaling for Elastic Applications," IEEE Internet Things J., vol. 7, no. 5, pp. 4195–4202, May 2020, doi:10.1109/JIOT.2020.2964405.

BG. M. Jin et al., "An Anomaly Detection Algorithm for Microservice Architecture Based on Robust Principal Component Analysis," IEEE Access, 2020, doi:10.1109/ACCESS.2020.3044610.

BH. C. K. Rudrabhatla, "A Quantitative Approach for Estimating the Scaling Thresholds and Step Policies in a Distributed Microservice Architecture," IEEE Access, vol. 8, pp. 180246–180254, 2020, doi:10.1109/ACCESS.2020.3028310.

BI. N. C. Coulson, S. Sotiriadis, and N. Bessis, "Adaptive Microservice Scaling for Elastic Applications," IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4195–4202, May 2020, doi:10.1109/JIOT.2020.2964405.

BJ. X. Guo et al., "Graph-Based Trace Analysis for Microservice Architecture Understanding and Problem Diagnosis," Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, vol. 20, 2020, doi:10.1145/3368089.

BK. D. Monteiro, P. H. M. Maia, L. S. Rocha, and N. C. Mendonça, "Building orchestrated microservice systems using declarative business processes," Service Oriented Computing and Applications, vol. 14, no. 4, pp. 243–268, Dec. 2020, doi:10.1007/S11761-020-00300-2/FIGURES/12.

BL. D. Ernst and S. Tai, "Offline Trace Generation for Microservice Observability," Proceedings—IEEE International Enterprise Distributed Object Computing Workshop, EDOCW, pp. 308–317, 2021, doi:10.1109/EDOCW52865.2021.00062.

BM. A. A. Khaleq and I. Ra, "Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications," IEEE Access, vol. 9, pp. 35464–35476, 2021, doi:10.1109/ACCESS.2021.3061890.

BN. A. Bento et al., "A layered framework for root cause diagnosis of microservices," 2021 IEEE 20th International Symposium on Network Computing and Applications (NCA), pp. 1–8, Nov. 2021, doi:10.1109/NCA53618.2021.9685494.

BO. T. Weng, W. Yang, G. Yu, P. Chen, J. Cui, and C. Zhang, "Kmon: An In-kernel Transparent Monitoring System for Microservice Systems with eBPF," Proceedings—2021 IEEE/ACM International Workshop on Cloud Intelligence, CloudIntelligence 2021, pp. 25–30, May 2021, doi:10.1109/CLOUDINTELLIGENCE52565.2021.00014.

BP. A. Megargel, C. M. Poskitt, and V. Shankararaman, "Microservices Orchestration vs. Choreography: A Decision Framework," pp. 134–141, Dec. 2021, doi:10.1109/EDOC52215.2021.00024.

BQ. Y. Chen, N. Chen, W. Xu, L. Lian, and H. Tu, "MFRL-CA: Microservice Fault Root Cause Location based on Correlation Analysis," pp. 90–101, Dec. 2021, doi:10.1109/DSA52907.2021.00018.

BR. Y. Cai, B. Han, J. Li, N. Zhao, and J. Su, "ModelCoder: A Fault Model based Automatic Root Cause Localization Framework for Microservice Systems," 2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS 2021, Jun. 2021, doi:10.1109/IWQOS52092.2021.9521318.

BS. M. Li, D. Tang, Z. Wen, and Y. Cheng, "Microservice Anomaly Detection Based on Tracing Data Using Semi-supervised Learning," 2021 4th International Conference on Artificial Intelligence and Big Data, ICAIBD 2021, pp. 38–44, May 2021, doi:10.1109/ICAIBD51990.2021.9459100.

BT. C. Pasomsup and Y. Limpiyakorn, "HT-RBAC: A Design of Role-based Access Control Model for Microservice Security Manager," pp. 177–181, Dec. 2021, doi:10.1109/BDEE52938.2021.00038.

BU. D. Liu et al., "MicroHECL: High-efficient root cause localization in large-scale microservice systems," Proceedings—International Conference on Software Engineering, pp. 338–347, May 2021, doi:10.1109/ICSE-SEIP52600.2021.00043.

BV. Z. Ye, P. Chen, and G. Yu, "T-Rank: A lightweight spectrum based fault localization approach for microservice systems," Proceedings—21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021, pp. 416–425, May 2021, doi:10.1109/CCGRID51090.2021.00051.

BW. S. Wang, Z. Ding, and C. Jiang, "Elastic scheduling for microservice applications in clouds," IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 1, pp. 98–115, Jan. 2021, doi:10.1109/TPDS.2020.3011979.

BX. L. Gu, D. Zeng, J. Hu, H. Jin, S. Guo, and A. Y. Zomaya, "Exploring layered container structure for cost efficient microservice deployment," Proceedings—IEEE INFOCOM, vol. 2021-May, May 2021, doi:10.1109/INFOCOM42981.2021.9488918.

BY. J. Grohmann et al., "SuanMing: Explainable Prediction of Performance Degradations in Microservice Applications," ICPE 2021—Proceedings of the ACM/SPEC International Conference on Performance Engineering, pp. 165–176, Apr. 2021, doi:10.1145/3427921.3450248.

BZ. B. Choi, J. Park, C. Lee, and D. Han, "pHPA: A Proactive Autoscaling Framework for Microservice Chain," 5th Asia-Pacific Workshop on Networking (APNet 2021), vol. 7, pp. 65–71, Jun. 2021, doi:10.1145/3469393.3469401.

CA. Y. Li, Y. Zhang, Z. Zhou, and L. Shen, "Intelligent flow control algorithm for microservice system," Cognitive Computation and Systems, vol. 3, no. 3, pp. 276–285, Sep. 2021, doi:10.1049/CCS2.12013.

CB. C. T. Joseph and K. Chandrasekaran, "Nature-inspired resource management and dynamic rescheduling of microservices in Cloud datacenters," Concurrency and Computation: Practice and Experience, vol. 33, no. 17, p. e6290, Sep. 2021, doi:10.1002/CPE.6290.

CC. M. Daoud, A. el Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. el Fazziki, "A multi-model based microservices identification approach," Journal of Systems Architecture, vol. 118, p. 102200, Sep. 2021, doi:10.1016/J.SYSARC.2021.102200.

CD. L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "A Kubernetes controller for managing the availability of elastic microservice based stateful applications," Journal of Systems and Software, vol. 175, p. 110924, May 2021, doi:10.1016/J.JSS.2021.110924.

CE. L. Meng, F. Ji, Y. Sun, and T. Wang, "Detecting anomalies in microservices with execution trace comparison," Future Generation Computer Systems, vol. 116, pp. 291–301, Mar. 2021, doi:10.1016/J.FUTURE.2020.10.040.

CF. V. Cortellessa, D. di Pompeo, R. Eramo, and M. Tucci, "A model-driven approach for continuous performance engineering in microservice-based systems," Journal of Systems and Software, vol. 183, p. 111084, Jan. 2022, doi:10.1016/J.JSS.2021.111084.

CG. J. Moeyersons, S. Kerkhove, T. Wauters, F. de Turck, and B. Volckaert, "Towards cloud-based unobtrusive monitoring in remote multi-vendor environments," Software: Practice and Experience, vol. 52, no. 2, pp. 427–442, Feb. 2022, doi:10.1002/SPE.3029.

Appendix B. Study Quality

	Reporting		Relevance		Rigor		Credibility		
	Aim	Scope, Context and Design	Implications in Practice and Research	Validity and Reliability of Variables	Explicitness of Measures	Adequacy of Reporting	Creditability, Validity and Reliability	Limitations	
Primary Study	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Total**
A	1.0	1.0	0.5	0.5	0.0	0.0	0.5	0.0	3.5
B	1.0	1.0	0.5	0.5	0.0	0.0	0.5	0.0	3.5
C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
D	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
E	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.0	6.5
F	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.0	3.5
G	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.0	6.5
H	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
I	1.0	1.0	0.5	0.5	0.0	0.0	0.5	0.0	3.5
J	1.0	1.0	1.0	0.5	0.0	0.0	1.0	1.0	5.5
K	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
L	0.5	1.0	1.0	1.0	0.5	1.0	1.0	0.0	6.0
M	0.5	0.5	1.0	1.0	1.0	1.0	1.0	0.0	6.0
N	1.0	1.0	1.0	1.0	0.5	1.0	1.0	0.0	6.5
O	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
P	1.0	1.0	0.5	0.0	0.0	0.0	0.5	0.5	3.5
Q	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
R	1.0	0.5	1.0	1.0	1.0	1.0	1.0	0.0	6.5
S	0.5	1.0	1.0	0.5	0.0	0.0	1.0	0.0	4.0
T	0.5	0.5	1.0	0.5	0.5	1.0	0.5	0.0	4.5
U	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	5.0
V	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	7.5
W	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
X	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
Y	1.0	1.0	1.0	0.5	0.0	0.0	1.0	1.0	5.5
Z	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
AA	0.5	1.0	1.0	1.0	0.5	1.0	1.0	0.0	6.0
AB	0.5	1.0	1.0	0.5	0.0	0.0	1.0	0.0	4.0
AC	1.0	1.0	1.0	0.5	0.0	0.0	0.5	0.0	4.0
AD	1.0	0.5	1.0	0.5	0.5	1.0	1.0	1.0	6.5
AE	1.0	1.0	1.0	1.0	0.5	1.0	1.0	0.0	6.5
AF	1.0	0.5	1.0	0.5	0.5	1.0	1.0	0.0	5.5
AG	1.0	1.0	0.5	0.5	0.0	0.0	0.5	0.0	3.5
AH	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	5.0
AI	1.0	1.0	1.0	0.5	0.0	1.0	0.5	1.0	6.0
AJ	1.0	0.5	0.5	0.5	0.0	0.0	1.0	1.0	4.5
AK	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
AL	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	4.0
AM	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
AN	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
AO	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
AP	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
AQ	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.5	5.5
AR	1.0	1.0	1.0	1.0	0.5	0.5	1.0	0.0	6.0

	Reporting		Relevance		Rigor		Credibility		
	Aim	Scope, Context and Design	Implications in Practice and Research	Validity and Reliability of Variables	Explicitness of Measures	Adequacy of Reporting	Creditability, Validity and Reliability	Limitations	
Primary Study	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Total**
AS	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
AT	1.0	1.0	1.0	1.0	0.5	0.5	1.0	1.0	7.0
AU	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
AV	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
AW	1.0	0.5	1.0	1.0	1.0	1.0	0.5	0.0	6.0
AX	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
AY	1.0	1.0	1.0	0.5	1.0	0.5	1.0	0.0	6.0
AZ	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
BA	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
BB	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
BC	1.0	1.0	1.0	0.5	1.0	1.0	1.0	0.5	7.0
BD	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.0	6.5
BE	1.0	1.0	1.0	1.0	0.0	0.0	0.5	0.0	4.5
BF	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
BG	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
BH	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
BI	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
BJ	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.0	4.5
BK	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
BL	1.0	1.0	0.5	0.5	0.5	0.5	0.5	1.0	5.5
BM	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
BN	1.0	0.5	0.5	0.5	0.5	0.5	0.5	0.0	4.0
BO	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.0	4.5
BP	1.0	1.0	1.0	0.5	0.5	0.5	0.5	0.0	5.0
BQ	1.0	1.0	0.5	1.0	1.0	1.0	1.0	0.0	6.5
BR	1.0	1.0	0.5	1.0	1.0	1.0	1.0	0.0	6.5
BS	1.0	1.0	0.5	0.5	1.0	1.0	1.0	0.5	6.5
BT	1.0	0.5	0.5	0.5	0.0	0.5	0.5	0.0	3.5
BU	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	7.5
BV	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
BW	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
BX	1.0	1.0	0.5	1.0	1.0	1.0	1.0	0.0	6.5
BY	1.0	1.0	0.5	1.0	1.0	1.0	1.0	1.0	7.5
BZ	1.0	0.5	0.5	0.5	0.5	0.5	0.5	0.0	4.0
CA	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
CB	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5
CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
CD	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
CE	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
CF	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
CG	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	7.5

References

1. Josuttis, N. *Soa in Practice: The Art of Distributed System Design*; O'Reilly Media, Inc.: Boston, MA, USA, 2007; ISBN 0596529554.
2. Fowler, M.; Lewis, J. Microservices. Available online: https://martinfowler.com/articles/microservices.html (accessed on 17 March 2022).
3. Newman, S. *Building Microservices*, 1st ed.; O'Reilly Media, Inc.: Boston, MA, USA, 2015; ISBN 1491950358, 9781491950357.
4. Evans, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*; Addison-Wesley: Reading, MA, USA, 2004.
5. Tekinerdogan, B.; Aksit, M. Classifying and evaluating architecture design methods. In *Software Architectures and Component Technology*; Springer: Boston, MA, USA, 2002; pp. 3–27.
6. Di Francesco, P.; Lago, P.; Malavolta, I. Architecting with Microservices: A Systematic Mapping Study. *J. Syst. Softw.* **2019**, *150*, 77–97. [CrossRef]
7. Villamizar, M.; Garcés, O.; Castro, H.; Verano, M.; Salamanca, L.; Casallas, R.; Gil, S. Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud. In Proceedings of the 2015 10th Computing Colombian Conference (10CCC), Bogota, Colombia, 21–25 September 2015; pp. 583–590.
8. Kitchenham, B.; Charters, S. Guidelines for Performing Systematic Literature Reviews in Software Engineering. *Engineering* **2007**, *2*, 1051–1052. [CrossRef]
9. O'Connor, R.V.; Elger, P.; Clarke, P.M. Continuous Software Engineering—A Microservices Architecture Perspective. *J. Softw. Evol. Process* **2017**, *29*, e1866. [CrossRef]
10. Zimmermann, O. Microservices Tenets. *Comput. Sci.* **2017**, *32*, 301–310. [CrossRef]
11. Shadija, D.; Rezai, M.; Hill, R. Towards an Understanding of Microservices. In Proceedings of the ICAC 2017—2017 23rd IEEE International Conference on Automation and Computing: Addressing Global Challenges through Automation and Computing, Huddersfield, UK, 7–8 September 2017. [CrossRef]
12. Benevides, R. Istio on Kubernetes. Available online: http://bit.ly/istio-kubernetes%0A (accessed on 17 March 2022).
13. Jamshidi, P.; Pahl, C.; Mendonca, N.C.; Lewis, J.; Tilkov, S. Microservices: The Journey so Far and Challenges Ahead. *IEEE Softw.* **2018**, *35*, 24–35. [CrossRef]
14. Pahl, C.; Brogi, A.; Soldani, J.; Jamshidi, P. Cloud Container Technologies: A State-of-the-Art Review. *IEEE Trans. Cloud Comput.* **2017**, *7*, 677–692. [CrossRef]
15. Kitchenham, B.; Brereton, O.P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic Literature Reviews in Software Engineering—A Systematic Literature Review. *Inf. Softw. Technol.* **2009**, *51*, 7–15. [CrossRef]
16. Pattern: Saga. Available online: https://microservices.io/patterns/data/saga.html (accessed on 17 March 2022).
17. Brewer, E. CAP Twelve Years Later: How the "Rules" Have Changed. *Computer* **2012**, *45*, 23–29. [CrossRef]
18. Dybå, T.; Dingsøyr, T. Strength of Evidence in Systematic Reviews in Software Engineering. In Proceedings of the 2008 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Kaiserslautern, Germany, 9–10 October 2008; ACM Press: New York, NY, USA, 2008; pp. 178–187.
19. Tyszberowicz, S.; Heinrich, R.; Liu, B.; Liu, Z. *Identifying Microservices Using Functional Decomposition*; Springer: Cham, Switzerland, 2018; pp. 50–65.
20. Tekinerdogan, B.; Bilir, S.; Abatlevi, C. Integrating Platform Selection Rules in the Model Driven Architecture Approach. In *Model Driven Architecture. Lecture Notes in Computer Science*; Aßmann, U., Aksit, M., Rensink, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3599.
21. Aksakalli, I.K.; Celik, T.; Can, A.B.; Tekinerdogan, B. A Model-Driven Architecture for Automated Deployment of Microservices. *Appl. Sci.* **2021**, *11*, 9617. [CrossRef]
22. Söylemez, M.; Tekinerdogan, B.; Kolukısa Tarhan, A. Feature-Driven Characterization of Microservice Architectures: A Survey of the State of the Practice. *Appl. Sci.* **2022**, *12*, 4424. [CrossRef]
23. 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. [CrossRef]
24. Soldani, J.; Andrew, D.; Van Den Heuvel, W.-J. The Journal of Systems and Software The Pains and Gains of Microservices: A Systematic Grey Literature Review. *J. Syst. Softw.* **2018**, *146*, 215–232. [CrossRef]
25. 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.
26. Vural, H.; Koyuncu, M.; Guney, S. *A Systematic Literature Review on Microservices*; Springer: Cham, Switzerland, 2017; pp. 203–217.
27. 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. [CrossRef]
28. Muhammad, W.; Liang, P.; Shahin, M. A systematic mapping study on microservices architecture in devops. *J. Syst. Softw.* **2020**, *170*, 110798.
29. Cerny, T.; Donahoo, M.J.; Trnka, M. Contextual understanding of microservice architecture: Current and future directions. *ACM SIGAPP Appl. Comput. Rev.* **2018**, *17*, 29–45. [CrossRef]
30. Dragoni, N.; Giallorenzo, S.; Lafuente, A.L.; Mazzara, M.; Montesi, F.; Mustafin, R.; Safina, L. Microservices: Yesterday, Today, and Tomorrow. In *Present and Ulterior Software Engineering*; Mazzara, M., Meyer, B., Eds.; Springer: Cham, Switzerland, 2017.