Highlights

Assessment of Performance and its Scalability in Microservice Architectures: Systematic Literature Review

Helena Rodrigues,António Rito Silva,Alberto Avritzer

- The paper answers the following research question: How does current research on microservice architectures, performance, and scalability address the architectural dimensions used in end-to-end scalability assessment?
- We **performed a systematic literature review** on the performance and scalability assessment of microservice architectures, where we screened 801 studies to select 29 primary studies that were used to provide an overview of the state of the art.
- We introduced a **systematic framework** for the analysis of performance and scalability assessment methodologies, and analyzed the 29 primary studies according to this analysis framework.
- We conclude that current research mainly focuses on some of the dimensions of the microservice architecture, namely the aspects associated with deployment architectures. In addition, it is difficult to compare the different studies because they address different aspects of the dimensions.
- We report a set of open research trends to guide researchers in the area of microservice-based software architecture quality assessment.

Assessment of Performance and its Scalability in Microservice Architectures: Systematic Literature Review

Helena Rodrigues[a,*], António Rito Silva[b] and Alberto Avritzer[c]

[a]Centro Algoritmi/LASI, University of Minho, Campus de Azurém, Av. da Universidade, Guimarães, 4800-058, Portugal
[b]INESC-ID, Instituto Superior Técnico, University of Lisbon, Rua Alves Redol 9, Lisboa, 1000-029, Portugal
[c]eSulabSolutions, Inc., Princeton, USA

ARTICLE INFO

Keywords:
Microservice Architecture
Performance
Scalability
Quality Assessment
Systematic Literature Review

ABSTRACT

The microservice architecture structures an application as a collection of small autonomous services, enhancing development practices and maintainability. It impacts system performance and scalability, which are influenced by the architectural design, the system deployment, and usage. In this paper, we present a systematic literature review (SLR) on the assessment of performance and scalability in the microservice architecture, aiming to identify research gaps and assess the current state of knowledge. The review protocol screened 801 studies, selecting 29 primary studies for detailed analysis. We perform comparisons of studies and try to identify the main gaps in current research and suggest areas for further investigation. The main conclusion is that the current research partially addresses the dimensions associated with the performance and scalability qualities of microservice systems. In particular, there is a lack of comparative studies of architectural patterns and styles or comparable assessment models and methods. Therefore, a systematic approach is lacking and the paper reports a set of open research trends to guide researchers in the area of microservice-based software architecture quality assessment.

1. Introduction

Microservice architecture is an architectural style that structures an application as a collection of small autonomous services, modeled around a business domain [1, 2, 3]. Each service can be developed independently using a cross-functional approach, and its modular structure can lead to better development practices and software that is easier to understand and maintain.

The architecture of microservices has an impact on the overall performance of the system and its scalability [4]. In particular, a survey reviewed existing research that compares the performance of monolithic systems to their microservice architecture equivalents [5], concluding that it is a multifaceted issue influenced by various factors, such as architectural design and implementation or operational profiles. However, this survey focused only on the impacts of migration to microservice architectures, not performing an extensive analysis of current research on performance and scalability assessment in microservice architectures.

Scalability can be characterized as architectural structure scalability or system load scalability. A system architecture is considered structurally scalable if its architecture design does not restrict the number of objects it can hold simultaneously (e.g., connections, transactions, or users) [6]. In contrast, a system is considered load scalable if it is able to satisfy performance requirements when the offered load is increased [7]. Offered load can be presented to the system by active connections, transactions, or users. In addition, elastic scaling is proposed to increase the architecture structure in response to an increase in load [8].

A performance and scalability end-to-end assessment approach of microservice architectures consists of the following activities: (1) the definition of the service-level objective; (2) the operational profile definition of load types and their probability of occurrence; (3) the evaluation of microservice architecture alternatives; and (4) the identification of the architecture microservice resources activated in the performance and scalability assessment.

Due to the diversity and complexity of these activities, in this paper, we present a systematic literature review (SLR) on the assessment of performance and its scalability in microservice architectures.

The aim of this study is to analyze if the current literature addresses the impact the microservice architecture has on system performance and scalability. This is particularly relevant, as the cost and complexity associated with microservice architecture have been reported to be prohibitive, and some organizations are backtracking and implementing the system again as a monolith [9]. This research question is further refined in Section 3, where we describe the literature review methodology. To our knowledge, there are no other systematic reviews published in the literature that focus on the aspects researched in this paper.

This systematic literature review has been carried out according to a review protocol adapted from [10]. The scope of the review includes the published literature from 2014 to the beginning of 2024. We have screened 801 studies and 29 of them were selected as primary studies that we used to address our research questions.

The key contributions of this paper are:

1. We **performed a systematic literature review** on the performance and scalability assessment of microservice architectures, where we screened 801 studies to select 29 primary studies that were used to provide an overview of the state of the art. The methodology used in this systematic literature review is presented in Section 3;

*Corresponding author

helena@dsi.uminho.pt (H. Rodrigues); rito.silva@tecnico.ulisboa.pt (A. Rito Silva); beto@esulabsolutions.com (A. Avritzer)

ORCID(s): 0000-0002-8978-8804 (H. Rodrigues); 0000-0001-9840-457X (A. Rito Silva); 0000-0002-9401-9663 (A. Avritzer)

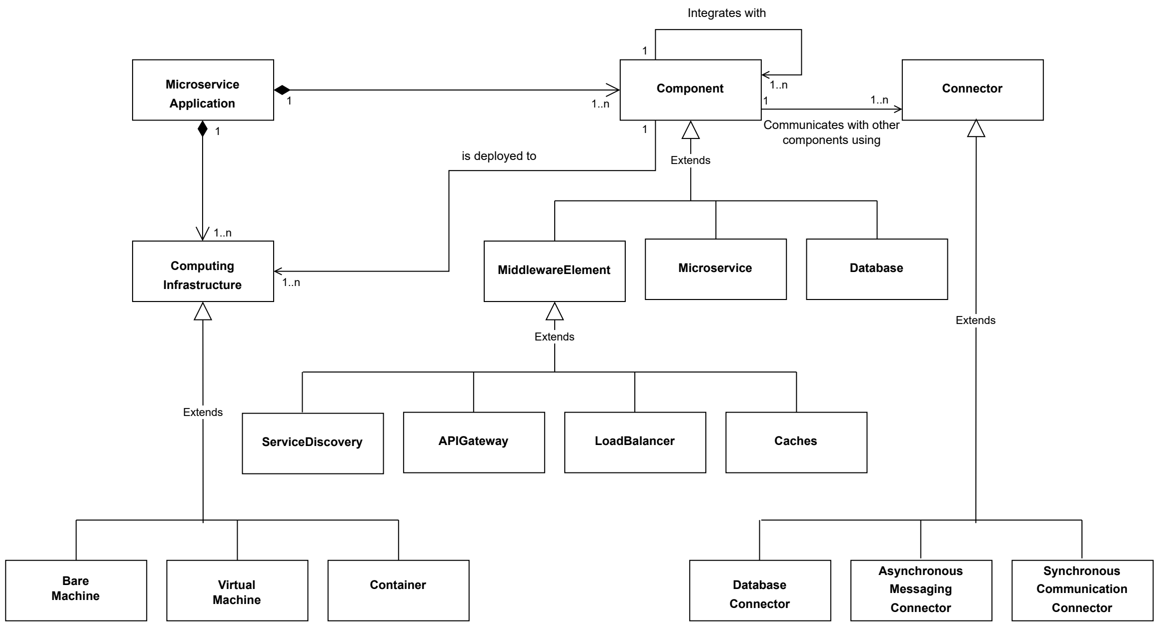

Figure 1: Microservice Architecture Architectural Elements

2. We introduced a **systematic framework** for the analysis of performance and scalability assessment methodologies, which is presented in Section 2, and analyzed the 29 primary studies according to this analysis framework. The results of this analysis are presented in Section 4;
3. We conclude that **current studies are incomplete**, in terms of the dimensions addressed, **incomparable**, in terms of how they realize each one of the dimensions, and, therefore, we have identified a **set of open trends** on the assessment process of the performance and scalability qualities in the microservice architecture, which are presented in Subsection 4.4.

The paper is structured as follows. In Section 2, we provide a detailed definition, with motivating examples, of the performance and scalability qualities analysis framework for the assessment of microservice architectures. The Analysis Framework is detailed in Subsection 2.1 with a reference model of the microservice architectural elements, and in Subsection 2.2 with the framework for the assessment of microservice architecture performance and scalability. Then, in Section 3, we present the literature review methodology, including the review protocol, the statement of the research objectives and questions (Subsection 3.1), the study selection (Subsection 3.2), the data extraction (Subsection 3.3), and the data synthesis (Subsection 3.4) procedures. The software architecture analysis of the identified studies is performed in Subsection 4.1, and the architecture quality assessment is done in Section 4.2. Sections 4.3 and 4.4 follow this analysis with a discussion of the results. Section 5 presents the identified threats to validity. Section 6 contains an overview of the related work on systematic literature review on microservice architectures. Finally, Section 7 contains our conclusions and guidance for future research.

2. Analysis Framework

In this paper, we focus on the dynamic aspects of microservice architectures performance and its scalability assessment methodologies. Following this objective, we first propose a conceptual framework to formally address these aspects and frame our analysis of selected primary studies. Therefore, this section is divided into two parts: 1) the microservice architectural designs (Section 2.1) and 2) the architecture quality assessment methodology (Section 2.2).

2.1. Microservice Architectural Designs

The process of adopting a microservice-based architecture is complex and influenced by numerous factors, such as microservices patterns, deployment and infrastructural patterns, microservices size, cohesion and decoupling, technical decisions, and consumer-driven contracts [1, 2, 3]. In this paper, we focus on the qualities of performance and scalability in microservice-based software architectures. To do so, we first introduce a reference model. This model will be the basis for our discussion of open issues concerning the impact of each of the elements of the reference model on the mentioned qualities. In addition, it will allow us to identify the extent to which each of the case studies analyzed addresses the different architectural aspects of microservice architecture.

Taking into account our focus on the dynamic aspects of microservice architecture, Figure 1 presents the architectural elements of the component-and-connector viewtype in microservice architectural views [11], which can influence the performance and scalability attributes.

In a component-and-connector view, the microservice system consists of components bound through connectors. The reference model considers 3 types of components: *Microservice*, *Database* and *Middleware*. The *Middleware* can correspond to *Service Discovery*, *API Gateway*, *Load Balancer*, and *Caches*. The components are bound through connectors: *Database* connector, *Asynchronous Messaging* connector, and *Synchronous Communication* connector.

Relevant for the analysis of the performance and scalability of the microservice architecture is how it is deployed in the communication and computing infrastructure. Therefore, it is important to describe the mapping between the components and connectors and the hardware of the computing platform on which the software is executed. The reference model also considers how components are deployed in *Bare Machines*, *Virtual Machines*, and *Containers*, which is another aspect that has a significant impact on the overall performance and scalability of microservice systems.

However, these architectural elements are applied using microservice patterns [12, 13], which combine them to provide the architectural properties required by microservice systems. In [14], different features that these patterns provide are identified. The following are relevant for the qualities of performance and scalability:

- **Data Management and Consistency**: Either the microservices share the database or there is a database per microservice. Correspond to the relations between *Microservice* and *Database* components in Figure 1 using the *Database* connector.
  When microservices have a **Shared Database** there are no data consistency issues because their changes are atomic, but if there is a **Database per Service**, data inconsistency can occur in the presence of distributed transactions. To cope with this situation, either a **Two Phase Commit Protocol** is followed, which supports ACID transactional behavior but does not scale as explained by the CAP theorem [15], or a **Saga** approach is followed, which supports eventual consistency through the orchestration, or choreography, of several local transactions that can rollback using compensating transactions [16, 13].
  When the **Database per Service** is followed, it is also necessary to manage the consistency of the query results, when the two phase commit protocol is not followed, because they are split between different microservices which may not be consistent. This can be addressed using patterns such as **API Composition**, which consolidates the outputs of various microservice queries, or **Command Query Responsibility Segregation (CQRS)**, which uses database replicas tailored for specific query types [13]. These two patterns are reported to have significant variations in terms of performance [13]. In fact, there is a quality trade-off between consistency and performance.

- **Communication Style**: Corresponds to the different styles and patterns that can be used in the communication between microservices and outside microservices.
  Communication Between Microservices: It is implemented by the *Synchronous Communication* connector and the *Asynchronous Messaging* connector in Figure 1. **Synchronous Communication** describes a microservice interaction where both microservices should be alive during the interaction, and the caller blocks waiting for the response from the callee. In **Asynchronous Messaging** the caller and the callee execute independently and the messages are stored in a messaging broker that is responsible to ensure communication reliability on microservices crashes.
  Communication Outside Microservices patterns describe whether there is **Direct Client to Microservice**, where the client has a direct access to the microservice, or there is an **API Gateway** that intermediates the communication between clients and microservices, providing common services, like authorization. It corresponds to one of the middleware components in Figure 1.

- **Service Orchestration**: Describes the dynamic behavior of the microservices associated with their coordination.
  The **Deployment** of microservices can follow different strategies, as illustrated in the computing infrastructure in Figure 1: **Bare Machine Based**; **Virtual Machine Based**; and **Container Based**.
  There are also microservice coordination strategies to react to changes in demand: **Horizontal**, where more microservice instances are added; and **Vertical**, where the capacity of the environment where the microservice is executed increases.
  Finally, **Load Balancing**, which redirects requests to microservice instances of the same microservice type, and **Service Discovery**, which hides the location of a microservice from its client, are used to decouple microservices and support autoscaling.

Both, the software architecture elements variations, and the features that organize those elements to provide quality to the solution, will be used in evaluation of the selected studies.

Note that some of these distinctions or separations are a bit blurred, due to the extensive use of development and deployment frameworks. For instance, if Spring Cloud[1] is used, an API Gateway is considered by default, and if Kubernetes[2] is used it is difficult to separate load balancing from service discovery. However, the studies analyzed consider these distinctions and so we decided to include them in the analysis framework.

2.2. Architecture Quality Assessment

In this section, we describe the architecture quality assessment framework, whose domains and dimensions frame our analysis of selected primary studies and provide to the readers insightful data for the discussion of how microservice architectures, and which architectural alternatives, impact applications performance and scalability qualities. That is, system scalability evaluation under several load levels and system performance evaluation and improvement for the expected production load level. The quality reference model considers the following domains(adapted from [17]):

1. the definition of the service level objective;
2. the operational profile definition of load types;
3. the quality assessment approach;
4. the test target.

In the following, we provide an example of the methodology for each of these assessment domains.

The **service level objective** defines how the results are evaluated. The definition of a performance metric objective

[1]https://spring.io/projects/spring-cloud
[2]https://kubernetes.io/

is often expressed as a latency, throughput, or CPU threshold [18]. For example, in [18] the performance requirement threshold Γ is expressed as $\Gamma = \mu + 3 \times \sigma$, where μ is the response time measurement and σ is the standard deviation, for a low load level measurement. The performance requirement condition RT is expressed as $RT < \Gamma$. On the other hand, the definition of a scalability service level objective is often translated into a scalability requirement as described in [19, 20, 7]. An example of a scalability requirement is the formal specification of the performance non-scalability likelihood (PNL) metric [7]. It considers the system's state probabilities for which the performance requirement is not met. The PNL metric can be used to support system scalability assessment as a function of offered load using an analytical performance model as shown in [7], or using load testing results from the system under test as described in [17].

The **operational profile definition** consists of estimating the probability of occurrence of each load level for a certain workload (e.g., number of transactions, number of simultaneous users, number of database connections) under study. An workload is also defined by the type of request that may significantly impact the performance of a microservices architecture. Therefore, it is also important to distinguish between requests that result in accesses to the database (create, read, update, and delete) or whether they are computationally intensive. The operational profile is used to answer two questions: (i) How do we identify the workload situations to test? (ii) How representative are the selected tests with respect to production usage? [17].

The **quality assessment approach** is supported by system performance measurements under varying load levels [17, 19]. For an automated assessment, each microservice scalability requirement is also defined. The scalability requirement is formulated using load testing results, for each microservice, for a defined low load level. Using the formally defined scalability requirement, pass/fail results can be automatically obtained for every microservice activated in the load test, and for each load level in the input domain. It can vary from a simple comparison to the performance requirement threshold [17] to more sophisticated approaches. For example, in [21], the methodology generates a list of pairs of satisfied/unsatisfied workload intensities/architecture configurations.

The **test target** can be the system source code itself or a model of it. The latter allows for a more abstract approach and a possible generalization of the results. Some examples of the use of system models for test targets are reported in the literature, such as, for example, the learned performance model [21], log model based on runtime information [22], and queueing network modeling [23].

The main difference between performance and scalability assessment is that the performance assessment can be implemented with a more detailed and richer set of workloads than the scalability assessment, because the performance assessment is executed at the expected production load level to validate a defined service level agreement. In contrast, scalability assessment is executed for a range of load levels and at a higher-level of abstraction.

3. Literature Survey Methodology

Having the background and rationale defined in Section 1, we must define the systematic literature review (SLR) protocol. A SLR protocol specifies the methods that will be used to perform a specific systematic review. Our review protocol includes the following stages, which were defined in [10]:

- The definition of research questions to be addressed;
- The definition of search and selection processes: the strategy that will be used to search for primary studies including search terms and resources to be searched;
- The definition of the study selection criteria and the evaluation of quality assessment scores: the protocol should describe how the selection criteria will be applied;
- Development of data extraction strategy: how the information required from each primary study will be obtained.
- Definition of data synthesis phase: defines the synthesis strategy.

3.1. Objective and research questions

The objective of this study is to summarize the state of the art in the assessment of scalability and performance of microservice architectures. In the context of this objective, the initial research question can be divided into the following two research questions:

Q1 What are the microservice architecture architectural elements, and respective variations, considered in the assessment of the performance and scalability qualities of microservice architectures?

Q2 Which are the service level objectives, operational profile definition, quality assessment approaches, test targets, and respective variations, used in the assessment of the performance and scalability qualities of microservice architectures?

3.2. Search, selection and quality assessment

In the study selection process, the objective is to generate a search strategy and select the relevant primary studies. Considerable work has been published on how to design a rigorous search strategy that maximizes the collection of relevant studies. In particular, the work in [24] points out the need for search strategies that optimize the retrieval of relevant articles from digital libraries and electronic databases. It proposes a systematic approach to search in order to improve the rigor of search processes in SLRs, which consists of five steps.

In the first step, we have chosen the venues where high-quality studies are published to conduct a manual search. In addition, these venues are also used in electronic search databases to later perform the automated search. The venues are listed in Table 1. To support automated search, we used Scopus. This is a comprehensive database of scholarly literature that is recognized as suitable for a variety of tasks, from journal and literature selection to large-scale bibliometric analysis [25]. It also overlaps heavily with the other major

Table 1
High-quality venues.

IEEE Transactions on Software Engineering
IEEE Transactions on Cloud Computing
Empirical Software Engineering
Journal of Systems Architecture
ACM Transactions on Software Engineering and Methodology
Journal of Systems and Software
Software: Practice and Experience
Information and Software Technology
International Conference on Software Engineering (ICSE)
European Software Engineering Conference (ESEC)
International Conference on Software Architectures (ICSA)
European Conference on Software Architectures (ECSA)
International Symposium on Empirical Software Engineering and Measurement (ESEM)
Measurement and Modeling of Computer Systems (SIGMETRICS)
Automated Software Engineering Conference (ASE)
IEEE International Conference on Software Maintenance and Evolution (ICSME)
International Conference on Evaluation and Assessment in Software Engineering (EASE)
IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)
International Symposium on Empirical Software Engineering (ISESE)
ACM/SPEC International Conference on Performance Engineering
IEEE International Conference on Software Services Engineering (IEEE SSE)

Table 2
Inclusion/Exclusion criteria.

Criteria type (Inclusion/Exclusion)	ID	Description
Inclusion (IC)	IC1	Title or abstract or keywords include key terms
	IC2	The abstract of the study indicates that the study is about microservices' architecture
	IC3	The study addresses scalability or performance assessment
	IC4	The study discusses the impact of architecture decisions on the evaluation metrics
	IC5	The study is written in the English language
	IC6	The study is published in Q1/Q2 journals or CORE A/B conferences
	IC7	The study does assess the scalability or performance of microservice architectural or deployment alternatives
Exclusion (EC)	EC1	Studies that do not explicitly relate to or discuss microservices
	EC2	Studies which main objective is not to assess scalability or performance of microservice architectural or deployment alternatives (ex: benchmark generators, placement algorithms, etc)
	EC3	Studies that discuss a specific application domain
	EC3	Documents which are conference proceedings, books, book chapters, or surveys.
	EC4	Studies with equivalent authors and that present equivalent results
	EC5	Studies where the full text is not available

bibliography indexing databases, including Web of Science (WoS) [26].

In the second step, we have performed a manual search in the selected venues before the search query is defined. The outcomes (primary studies) of the manual search create a *Quasi-gold standard* (QGS), which is subsequently utilized to derive the search strings for the automated search [24]. We analyzed the period between 2014 and the beginning of 2024.

The manual search was conducted by screening all articles published in the selected publication venues during the defined time frame. The title-abstract-keywords fields of each primary study were analyzed to identify those studies that could provide evidence about our research questions. Studies that mentioned the term microservices were selected and then further analyzed (this means that the complete text was analyzed) for the application of inclusion (IC) and exclusion criteria (EC).

IC and EC were defined during the protocol definition, prior to manual and automated search execution. The inclusion (IC) and exclusion criteria (EC) that we have defined are listed in Table 2.

The manual search on specific venues resulted in 37 selected studies. Then, 23 studies were eliminated because they did not address our research questions or meet the inclusion criteria. As a result, 14 primary studies are selected to analyze and produce the search query strings.

The third step consists of the definition of search strings. We used a subjective search string definition [24]. We have defined a complex search string to perform an automated search. The keywords have been selected according to the research questions we want to address, researchers domain knowledge and past experiences, combined with elicitation of the recommended search terms and phrases extracted from title-abstract-keywords from relevant studies on the QGS (list of papers retrieved from manual search, in the second step). The resulting search strings are presented in Table 3.

The fourth step consists of conducting the automated search. The selected digital library, Scopus, is searched using the proposed search string. Due to overlapping between digital sources, duplicate papers retrieved from manual search and Scopus database search are identified and removed in this step.

Based on the selected keywords, 801 unique papers are identified using the automatic database search method. Then

Table 3
Search strings.

(((TITLE-ABS-KEY ("software architecture") OR TITLE-ABS-KEY ("microservice* architecture*") OR TITLE-ABS-KEY ("microservice-based architecture*") OR TITLE-ABS-KEY ("microservice* pattern*") OR TITLE-ABS-KEY ("deployment architecture") OR TITLE-ABS-KEY ("architecture deployment") OR TITLE-ABS-KEY ("microservice* system*") OR TITLE-ABS-KEY ("microservice-based system*")) AND (TITLE-ABS-KEY (microservice*) OR TITLE-ABS-KEY (microservice*)) AND (TITLE-ABS-KEY (assess*) OR TITLE-ABS-KEY (test*) OR TITLE-ABS-KEY (evaluati*) OR TITLE-ABS-KEY (benchmark*) OR TITLE-ABS-KEY (model*) OR TITLE-ABS-KEY (metrics) OR TITLE-ABS-KEY ("operational profile") OR TITLE-ABS-KEY (workload)) AND (TITLE-ABS-KEY (performance) OR TITLE-ABS-KEY (scalab*))) AND PUBYEAR > 2013)

Table 4
Quality checklist for primary studies.

Quality metric (question)	Description
Q1	Are the dimensions used in the study likely to be valid and reliable?
Q2	Is the research process documented adequately?
Q3	Are all the study questions answered?

11 papers are eliminated because they match the manually selected papers. As a result, 15 primary studies are selected by applying the inclusion and exclusion criteria in table 2 to the 790 studies obtained with the automated search, resulting in a total of 29 primary studies.

During this phase, we have also performed a quality check adapted from the quality assessment process in [10]. Primary studies were also evaluated according to the quality checklist presented in Table 4. In this way, we have also eliminated studies that do not present sound results as a whole (if they are too short or use toy examples).

The complete study selection process is represented in Figure 2 and the selected studies are presented in tables 5 and 6.

In the fifth and final step, search performance is evaluated against QGS to ensure automated search quality. As proposed in [24], we use the quasi-gold standard (from manually selected venues) to measure a quasi-sensitivity value. We calculated the number of relevant papers retrieved from the selected venues (Step 1) through automated search (Step 4). This number must not be greater than the number of papers identified in Step 2. The corresponding 'quasi-sensitivity' is calculated by dividing this number by the QGS pool size. Figure 3 illustrates the result composition of the initial and final searches and the contributions of the manual search (QGS) and the automated search. The 'quasi-sensitivity' value of our search strategy is 80% (11/14), which is considered acceptable [24], and the results of the automated search can be merged with QGS, and the search process is ended.

3.3. Data extraction

We read the full text of the primary studies selected for data collection. In our work, we specify different dimensions that address our research questions to effectively and efficiently extract the data from the selected primary studies. These dimensions are defined according to the variations in the architectural reference model and the quality architecture assessment described in Sections 2.1 and 2.2.

In our work, at least two researchers reviewed each primary study, as recommended in reviews of the software literature.

3.4. Data synthesis

Data synthesis involves collating and summarizing the approaches of the included primary studies. This process summarizes and synthesizes the data extracted from the primary studies to answer research questions.

In this phase, the primary objective is to classify, analyze, and understand current research addressing the assessment of microservice architecture performance and scalability and to what extent and how architectural issues are considered in this assessment. We perform a descriptive analysis of the content of the primary studies and describe how each study addresses the main dimensions we have identified as relevant to architecture, scalability, and performance assessment in microservices systems. These analyses are described in Sections 4.1 and 4.2.

Following these activities, further qualitative analysis is carried out on the data obtained as a result of the analysis of the set of studies as a whole. We perform comparisons of studies and try to identify the main gaps in the scalability and performance assessment of microservice systems in the context of microservices architectures. This discussion is presented in Section 4.3.

4. Analysis

In this section, results relevant to our research questions are extracted. For each research question, the data extracted from the selected studies are presented in a separate subsection. In Section 4.1 we synthesize data on microservices architecture designs; In Section 4.2 we synthesize architecture quality assessment; In Section 4.3 we present our discussion on the state-of-the-art; and in Section 4.4 we identify a set of open trends on the assessment process of the performance and scalability qualities in the microservice architecture.

4.1. Microservice Architecture Designs

[Q1] What are the architectural elements of the microservice architecture, and the respective variations, considered in evaluating the performance and scalability qualities of microservice architectures?

The selected articles cover different aspects of the architectural elements and concerns identified in Section 2.1. In the analysis performed, we distinguish whether the architectural element was used in the *design* of the experiment or whether it was also tested with several *variations*. While the latter allow us to assess the impact on performance and scalability due to the presence of the architectural element, the former does not because it is not possible to compare.

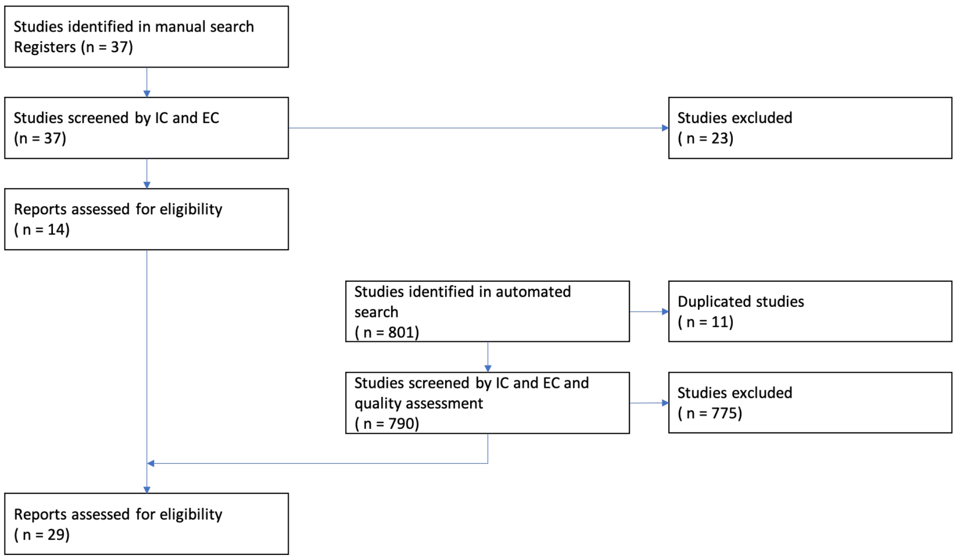
Studies identified in manual search Registers (n = 37)
Studies screened by IC and EC (n = 37)
Studies excluded (n = 23)
Reports assessed for eligibility (n = 14)
Studies identified in automated search (n = 801)
Duplicated studies (n = 11)
Studies screened by IC and EC and quality assessment (n = 790)
Studies excluded (n = 775)
Reports assessed for eligibility (n = 29)

Figure 2: Study selection process

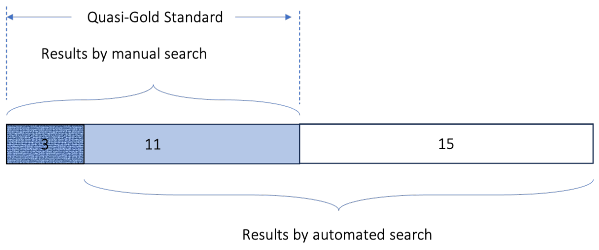

Figure 3: Results from manual and automated search

The studies thoroughly explore variations in **deployment and scaling** (Table 7). In particular, P2, P3, P5, P8, S1, S2, S6, S11, and S13-S15 (11 in total) exercise several deployment variations of public clouds (Google Cloud Platform, Oracle Cloud Infrastructure, Azure Spring Cloud and Azure App Service) and private clouds, technologies (Spring Cloud and Kubernetes), assignment of microservices to virtual machines and containers, bare metal, or virtualization. On the other hand, P3-P9, P12, S1, S2, S6-S8, S11-S17 (20 in total) analyze variations of horizontal scaling, while P4, P5, P7, P8, S2, S6, S7, and S15 (8 in total) study the impact of vertical scaling on performance and scalability. Therefore, we notice that more studies focus on horizontal scaling than vertical scaling. In general, scaling, horizontally or vertically, is the main architectural concern (20/29) addressed by the studies.

Interestingly, although deployment and scaling are the architectural aspects more thoroughly addressed, there are few studies that analyze variations in **service orchestration's architectural components**: load balancing and service discovery. There are 16 studies (P3-P4, P7-P9, P11, P12, S2, S3, S6-S8, S12-S14, S16) that use a load balancer, as a design element, in the experiment, but only 3 studies actually evaluate its impact on performance and scalability: P12 evaluates the quality, comparing with and without the load balancer; S13 compares the location of the load balancer on the client side, server side, and a hybrid solution; and S14 compares Netflix Ribbon with Kubernetes Service. In terms of the service discovery component, 12 studies (P4, P7-P9, P11, S2, S3, S6-S8, S12, S14) include the service discovery component in their architecture, but only 1 evaluates its presence through variations: S14 compares Netflix Eureka with Kubernetes Service.

A similar situation can be observed in terms of the **communication style** in Table 8, where only a few experiments actually study the impact of the software architectural element on the quality. Therefore, concerning the communication between microservices, 9 studies only use independent microservices, which do not intercommunicate; in 13 studies (P4, P6-P9, P11, S2, S4, S5, S7, S8, S12, S14) the communication is synchronous; in 5 (P10, S1, S11, S15, S17) is asynchronous; and only 1 study (S16) has an experiment that includes both types of communication. However, of the 19 studies that have inter-microservice communication, only 4 analyze the impact of that communication on performance and scalability: P6 studies variations in synchronous communication to reduce dependency cycles between microservices; S11 examines Kafka Streams and Flink in asynchronous communication; S15 explores the differences in messaging systems; and S17 compares the use of polling- and push-based brokers. The same pattern can be observed with communication outside services, where P10 discusses variations of using aggregation and offloading in an API Gateway, P12 experiments with an API Gateway that requests bundle and rate limit, S9 compares an API Gateway in REST and GraphQl, and S14 studies implementation

Table 5
Performance Assessment Selected Studies.

Study number	Authors	Title	Year	Source
P1	Catia Trubiani, Anne Koziolek, Vittorio Cortellessa, Ralf Reussner [27]	Guilt-based handling of software performance antipatterns in palladio architectural models	2014	Journal of Systems and Software
P2	Amaral M.; Polo J.; Carrera D.; Mohomed I.; Unuvar M.; Steinder M. [28]	Performance evaluation of microservices architectures using containers	2016	IEEE International Symposium on Network Computing and Applications
P3	Villamizar M.; Garcés O.; Ochoa L.; Castro H.; Salamanca L.; Verano M.; Casallas R.; Gil S.; Valencia C.; Zambrano A.; Lang M. [29]	Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures	2017	Service Oriented Computing and Applications
P4	Eismann S.; Bezemer C.-P.; Shang W.; Okanović D.; Van Hoorn A. [30]	Microservices: A performance tester's dream or nightmare?	2020	ACM/SPEC International Conference on Performance Engineering
P5	Jha D.N.; Garg S.; Jayaraman P.P.; Buyya R.; Li Z.; Morgan G.; Ranjan R. [31]	A study on the evaluation of HPC microservices in containerized environment	2021	Concurrency and Computation: Practice and Experience
P6	Liu L.; Tu Z.; He X.; Xu X.; Wang Z. [32]	An Empirical Study on Underlying Correlations between Runtime Performance Deficiencies and 'Bad Smells' of Microservice Systems	2021	IEEE International Conference on Web Services
P7	Camilli M.; Janes A.; Russo B. [21]	Automated test-based learning and verification of performance models for microservices systems	2022	Journal of Systems and Software
P8	Camilli M.; Russo B. [18]	Modeling Performance of Microservices Systems with Growth Theory	2022	Empirical Software Engineering
P9	Cortellessa V.; Di Pompeo D.; Eramo R.; Tucci M. [22]	A model-driven approach for continuous performance engineering in microservice-based systems	2022	Journal of Systems and Software
P10	Pinciroli R.; Aleti A.; Trubiani C. [23]	Performance Modeling and Analysis of Design Patterns for Microservice Systems	2023	IEEE International Conference on Software Architecture
P11	Tariq S.S.M.; Menard L.; Su P.; Roy P. [33]	MicroProf: Code-level Attribution of Unnecessary Data Transfer in Microservice Applications	2023	ACM Transactions on Architecture and Code Optimization
P12	Malki A.E.; Zdun U. [34]	Combining API Patterns in Microservice Architectures: Performance and Reliability Analysis	2023	IEEE International Conference on Web Services

of the API Gateway (Netflix Zuul, Spring Cloud Gateway, and Ingress). In addition to these 4 studies, only 5 other studies (P3, S6, S12, S16, S17) had an API Gateway in their experiments but did not analyze their impact. They are marked as design elements in the table.

The architectural aspects less addressed by the studies were the ones related to **data management and consistency**, as shown in Table 9. Only 2 studies considered the use of a shared database. Study P4 uses caches of the shared database in the microservices, whereas Study P6 discusses the impact of using a shared database on performance. A single study addressed data consistency, S13, by using a SAGA choreography. Concerning queries, only 4 studies discuss variations: P10 compares API Composition with CQRS; S4 discusses implementations of API Composition using choreography, orchestration and caches; S9 compares request aggregation and schema stitching in the API Composition; and S12 applies CQRS, replicated databases, and sharding. Another study (S16) uses API Composition but does not study its impact.

Finally, Table 10 presents the codebases that are used for benchmarking. Some codebases are repeatedly used in the studies: TrainTicket is used in 5 (P6, P9 P11, S12, S13), SockShop in 5 (P7, P8, S2, S7, S13) and DayTrader in 2 (P11, S5). Then the complexity of the benchmarks varies significantly, where some studies use microbenchmarks whose concern is to test the load of particular elements of the architecture, e.g. the asynchronous queues (P5, S1, S4, S11, S15) or CPU load (P2), or the microservice system is a set of simple independent, non inter-communicating, microservices

Table 6
Scalability Assessment Selected Studies.

Study number	Authors	Title	Year	Source
S1	M Gotin, F Lösch, R Heinrich, R Reussner [35]	Investigating Performance Metrics for Scaling Microservices in Cloud IoT-Environments	2018	ACM/SPEC International Conference on Performance Engineering
S2	Avritzer A.; Ferme V.; Janes A.; Russo B.; Hoorn A.V.; Schulz H.; Menasché D.; Rufino V. [17]	Scalability Assessment of Microservice Architecture Deployment Configurations: A Domain-based Approach Leveraging Operational Profiles and Load Tests	2020	Journal of Systems and Software
S3	Tapia F.; Mora M.A.; Fuertes W.; Aules H.; Flores E.; Toulkeridis T. [36]	From monolithic systems to microservices: A comparative study of performance	2020	Applied Sciences (Switzerland)
S4	Jayasinghe M.; Chathurangani J.; Kuruppu G.; Tennage P.; Perera S. [37]	An analysis of throughput and latency behaviours under microservice decomposition	2020	International Conference on Web Engineering
S5	Nitin V.; Asthana S.; Ray B.; Krishna R. [38]	CARGO: AI-Guided Dependency Analysis for Migrating Monolithic Applications to Microservices Architecture	2022	ACM International Conference Proceeding Series
S6	Blinowski G.; Ojdowska A.; Przybylek A. [39]	Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation	2022	IEEE Access
S7	Xu M.; Song C.; Ilager S.; Gill S.S.; Zhao J.; Ye K.; Xu C. [40]	CoScal: Multifaceted Scaling of Micro-services With Reinforcement Learning	2022	IEEE Transactions on Network and Service Management
S8	Mulahuwaish A.; Korbel S.; Qolomany B. [41]	Improving datacenter utilization through containerized service-based architecture	2022	Journal of Cloud Computing
S9	Vohra N.; Kerthyayana Manuaba I.B. [42]	Implementation of REST API vs GraphQL in Microservice Architecture	2022	International Conference on Information Management and Technology
S10	Hassan S.; Bahsoon R.; Buyya R. [20]	Systematic scalability analysis for micro-services granularity adaptation design decisions	2022	Software - Practice and Experience
S11	Sören Henning; Wilhelm Hasselbring [43]	A configurable method for benchmarking scalability of cloud-native applications	2022	Empirical Software Engineering
S12	Camilli M.; Colarusso C.; Russo B.; Zimeo E. [19]	Actor-Driven Decomposition of Micro-services through Multi-level Scalability Assessment	2023	ACM Transactions on Software Engineering and Methodology
S13	Filippone G.; Pompilio C.; Autili M.; Tivoli M. [44]	An architectural style for scalable choreography-based microservice-oriented distributed systems	2023	Computing
S14	Wang Y.-T.; Ma S.-P.; Lai Y.-J.; Liang Y.-C. [45]	Qualitative and quantitative comparison of Spring Cloud and Kubernetes in migrating from a monolithic to a micro-service architecture	2023	Service Oriented Computing and Applications
S15	Henning S.; Hasselbring W. [46]	Benchmarking scalability of stream processing frameworks deployed as micro-services in the cloud	2024	Journal of Systems and Software
S16	Faustino D.; Gonçalves N.; Portela M.; Rito Silva A. [4]	Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation	2024	Performance Evaluation
S17	Batista C.; Morais F.; Cavalcante E.; Batista T.; Proença B.; Rodrigues Cavalcante W.B. [47]	Managing asynchronous workloads in a multi-tenant microservice enterprise environment	2024	Software - Practice and Experience

(P3, S3, S4, S6, S9). Of course, some studies use complex, sometimes real, microservice systems as a benchmark (P4, P8, P9, P11, P12, S7, S8, S12, S14, S16, S17). Note that the reused benchmarks have inter-microservice communication, and as such, they are also considered complex.

Overall, as result of the analysis in Tables 7 to 10, we observe that in most of the studies the key architectural

Table 7
Service Orchestration.

Service Orchestration	Application	Found in Study
Deployment	Variations	P2, P3, P5, P8, S1, S2, S6, S11, S13, S14, S15
	Design	P4, P6, P7, P9, P11, P12, S3, S4, S5, S7, S8, S9, S10, S12, S16, S17
	Not applied	P1, P10
Scaling	Vertical Variations	P4, P5, P7, P8, S2, S6, S7, S15
	Horizontal Variations	P3, P4, P5, P6, P7, P8, P9, P12, S1, S2, S6, S7, S8, S11, S12 S13, S14, S15, S16, S17
	Not applied	P1, P2, P10, P11, S3, S4, S5, S9, S10
Load Balancing	Variations	P12, S13, S14
	Design	P3, P4, P7, P8, P9, P11, S2, S3, S6, S7, S8, S12, S16
	Not applied	P1, P2, P5, P6, P10, S1, S4, S5, S9, S10, S11, S15, S17
Service Discovery	Variations	P12, S14
	Design	P4, P7, P8, P9, P11, S2, S3, S6, S7, S8, S12
	Not applied	P1, P2, P3, P5, P6, P10, S1, S4, S5, S9, S10, S11, S13, S15, S16, S17

Table 8
Communication Style.

Communication Style	Application	Found in Study
Synchronous Between Microservices	Variations	P6
	Design	P4, P7, P8, P9, P11, S2, S4, S5, S7, S8, S12, S14, S16
	Not applied	P1, P2, P3, P5, P10, P12, S1, S3, S6, S9, S10, S11, S13, S15, S17
Asynchronous Between Microservices	Variations	S11, S15, S17
	Design	P10, S1, S16
	Not applied	P1, P2, P3, P4, P5, P6, P7, P8, P9, P11, P12, S2, S3, S4, S5, S6, S7, S8, S9, S10, S12, S13, S14
API Gateway Outside Microservices	Variations	P10, P12, S9, S14
	Design	P3, S6, S12, S16, S17
	Not applied	P1, P2, P4, P5, P6, P7, P8, P9, P11, S1, S2, S3, S4, S5, S7, S8, S10, S11, S13, S15

Table 9
Data Management and Consistency.

Data Management and Consistency	Application	Found in Study
Shared Database	Variations	P6
	Design	P4
Data Consistency	Variations	S13
Query	Variations	P10, S4, S9, S12
	Design	S16

Table 10
Benchmarks.

Benchmark	Found in Study
TrainTicket	P6, P9, P11, S12, S13
SockShop	P7, P8, S2, S7, S13
DayTrader	P11, S5
Complex System	P4, P8, P9, P11, P12, S7, S8, S12, S14, S16, S17
Simple System	P3, S3, S4, S6, S9
Microbenchmarks	P2, P5, S1, S4, S11, S15
Not applied	P1, P10, S10

Table 11
Metric objectives found in selected studies.

Performance metric objective	Found in Study
Response time	P1, P2, P3, P4, P6, P8, P9, P10, P11, P12, S3, S5, S7, S8, S9, S10, S13, S16
Throughput	P9, P10, S4, S5, S6, S8, S9, S10, S14, S16
CPU utilization	P4, P5, P6, P9, P10, S1, S3, S8
Memory	P5, P6
IO and Network	P5
Continuous stochastic metric	P7
Message Queue's metrics based	S1
Probabilistic Domain-based metric	S2, S12
Connection time	S7
Successful requests	S7, S14
Errors	S8, S9
Lag trend (load x Resource space)	S11, S15
Scalability Gap	S12
Scalability Footprint	S12
Waiting time	S17

elements that are selected for performance and scalability assessment are the ones associated to deployment and scaling alternatives. We also observe that variations of the service orchestration's architectural elements, such as Service Discovery or Load Balancer are often considered in the selected studies, on one hand, but few studies actually evaluate the impact of such elements on performance and scalability, on the other hand. The less addressed aspects are those related to data management and its consistency, as well as the variations of the type of communication between services and outside services. The lack of analysis of these two aspects is particularly relevant because distributed transactions (coordination) are considered one of the hardest parts of microservice architecture, and asynchronous communication is considered essential for scalability, which is scarcely addressed in studies [48]. Additionally, it is also interesting to observe that the chosen codebases for benchmarking are not justified in terms of their extensive coverage, in terms of the architectural elements and their variations, but the main claim is that they are like a real system, which in most of the cases they are not. This means that they do not have the necessary requirements to be a benchmark, as defined in [49, 50], and so, they are closer to a reference microservice application, although the word benchmarking is recurrently used, e.g. [51].

4.2. Architecture Quality Assessment

[Q2] Which are the service level objectives, operational profile definition, quality assessment approaches, test targets, and respective variations, used in the assessment of the performance and scalability qualities of microservice architectures?

The selected studies cover different aspects of the domains and dimensions of the architecture quality assessment identified in Section 2.2.

Table 11 summarizes the dimensions found in the selected studies for the **definition of the service level objective (SLMO)** domain. The response time was used evenly in performance and scalability studies, while the throughput was used mainly in scalability studies. However, resource utilization metrics were prevalent in performance studies. Paper P7 uses continuous stochastic logic to measure the performance of the sequence of requests that are executed to fulfill a functionality, which captures an intrinsic property of microservice systems: microservice coordination. However, scalability studies used a more diverse set of metrics to capture the impact of lack of scalability, such as the number of errors (S8, S9), the waiting time (S17), and the scalability gap and footprint (S12). It also considers metrics associated with the use of message brokers in communication, which is a fundamental architectural component when there are scalability requirements. Study S1 uses message queue delay, message queue growth, and message queue length, and studies S11 and S15 use the lag trend metric that estimates the rate of change in message queue length [43]. The use of probabilistic metrics (S2, S12) in scalability reflects the ability of the architecture deployment configuration to satisfy the scalability requirement under a given operational profile. Due to autoscaling solutions for scalability, Study S7 uses connection time, response time, and successful requests under different scaling techniques (horizontal scaling, vertical scaling, and brownout), for the same workload. Some scalability studies also address resource utilization (S11, S15), but associate it with load intensity. Overall, we can observe that there is more diversity of metrics associated with scalability.

Table 12 summarizes the dimensions found in the selected studies for the **operational profile definition (OPD)** domain. Two main variations can be identified, and they depend on whether they use real system data to define the profile. When the OPD is inferred from system data: P8 uses a production-based operational profile estimation; in P9 production operational profile is analyzed at runtime; in S2 it is obtained based on the analysis of two production systems (a video streaming application and Wikipedia); S7 derives from a workload trace based on monitoring; and S12 uses automated analysis of historical data. When OPD is not

Table 12
Operational profile definition dimension found in selected studies.

Operational Profile Definition	Found in Study
Production	P8, P9, S2, S7, S12
Operations impact	S6, S9, S16
Model-specific	P7
Architecture-specific	S11, S15
Benchmarks	P5
Ad-hoc	P1, P2, P3, P4, P6, P10, P11, P12, S1, S3, S4, S5, S8, S13, S14, S17
Not Applicable	S10

Table 13
Quality Assessment Approach dimension found in selected studies.

Assessment Approach	Found in Study
Workload	P1, P3, P7, P8, S1, S3, S4, S5, S8, S9, S11, S13, S14, S15, S16, S17
Computation	P2, P3, P4, S1
Resources	P2, P4, P7, P8, S2, S6, S11, S12, S15
Architecture	P2, P4, P5, P6, P7, P8, P9, P10, P11, P12,
	S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S12 S13, S14, S15, S16, S17

inferred from real data it can be ad-hoc (P1-P4, P6, P10, P11, P12, S1, S3, S4, S5, S8, S13, S14, S17), where the authors do not present a rationale for OPD. It can also be defined based on the expected impact due to the type of operation, such as S16 (number of domain entities, number of microservices, processing time), S6 (query and data intensive operations), S9 (POST, PUT, and GET REST operations). In one study, P7, a model is used, a discrete-time Markov chain that models the use cases of the system. In some other approaches, benchmarks are used, like in P5. More specifically, the OPD can be defined in order to explore the architectural aspects, like in S11 and S15 which simulate messages generated by sensors to explore message brokers properties. There are no significant differences on how OPD is defined when comparing performance and scalability studies, although the latter have a greater concern about load levels. In terms of architectural aspects, few studies considered particular architectural aspects when defining OPD.

Table 13 summarizes the dimensions found in the selected studies for the **quality assessment approach** (SA) domain. The different approaches assess quality by comparing different types of variation: workload; computation; resources; and architecture. The workload variations analyze the performance and scalability quality of the systems depending on the workload intensity. The computation variations look at the impact of the type of operation, whether it is computationally intensive, read, or write database accesses. The resources variations look at the impact of CPU, memory and network capabilities. Finally, the architecture variations experiment with different architectural configuration, be it deployment configurations or logical configurations, like the use of different microservices architectural patterns.

The **test target** dimension is summarized in Table 14. Most studies perform the tests on the system under test. The few exceptions occur when there is concern about the design of a new system or the refactoring of the old one (P8, P9, P12, S3, S4), where the model is used to reasoning about the changes to perform. Exceptions are studies P1, P10 and S10, where the models are created a priori to reasoning about a set of patterns, in the former, and to do requirements analysis, in the latter.

Overall, we can verify that there is a high variety of metrics to assess scalability, whereas the set of metrics for performance assessment look more stable (Table 11). In terms of operational profile, there is a diversity of approaches, including a large number of studies that follow the definition of ad hoc operational profile (Table 12). Moreover, only in 2 studies the definition of the operational profile was driven by the architecture design; it is not clear whether they are complete in terms of having a good coverage of the architectural aspects; in general, the use cases generated use a black-box approach and therefore they do not cover the architecture elements. However, note that the assessment approach actually performs variations of the architecture (Table 13), but it was mainly on the deployment aspects. On the other hand, a lack of formalization can be observed in assessment approaches (Table 14), where the target of the tests is a system, instead of a model.

4.3. Discussion on the State of the Art

In this subsection, we discuss the main goals of the addressed primary studies that comprise the state-of-the-art on performance and scalability assessment.

Although the studies address the assessment of performance and scalability qualities in microservice architecture, the goals of each study are diverse. Tables 15, 16, 17 and 18 summarize the goals and results of each of the studies, while also focusing on the variations explored. In terms of its objectives, studies can be distinguished in different categories:

- Studies that perform a classical analysis of the qualities of a particular system or system architecture and

Table 14
Test Target dimension found in selected studies.

Test Target (TT)	Found in Study
Annotated UML and Log models	P9
KOAS goal oriented modeling	S10
Queuing Networks model	P1, P10, S4
Growth Theory model	P8
Regression model	P12, S3
System	P2, P3, P4, P5, P6, P7, P8, P9, P11,
	S1, S2, S3, S4, S5, S6, S7, S8, S9, S11, S12, S13, S14, S15, S16, S17

Table 15
Goals, variations and results of studies P1-P6

Study	**Goal**	**Variations**	**Results**
P1	How can performance antipatterns be identified and solved?	Architectural design alternatives.	Propose a ranking methodology that identifies, among a set of detected antipatterns, the antipatterns that more likely contribute to the violation of specific performance requirements.
P2	What is the performance impact of using master-slave or nested-container in the implementation of the microservice architecture, when compared with bare metal and virtual machine?	Deployment strategies (bare metal vs regular containers vs nested-containers vs VMs).	All environments display a similar behavior; no significant performance impact for CPU-intensive executions when running on containers or virtual machines compared to bare-metal.
P3	What is the cost variations of monolith vs. microservice-based architecture, considering different deployment scenarios?	Deployment startegies: monolithic architecture, microservice architecture operated by the cloud customer, and microservice architecture operated by the cloud provider.	Microservices deployments managed by cloud customers can reduce infrastructure costs but come with higher latency compared to monolithic deployments. Serverless deployments further reduce infrastructure costs while maintaining latency levels similar to monolithic deployments.
P4	How stable is the test execution environment?	Horizontal and vertical autoscaling; load variations on the number of simultaneous requests.	It is possible to detect performance regressions in microservices applications. Autoscalers have no deterministic behavior.
P5	What is the performance impact of microservices deployed in the intra-container or the inter-container environments?	Container-base (variations in terms of intra and inter container microservice interference).	The performance quality given intra and inter container deployment variations depends on microservices resource requirements and type of operations.
P6	What is the performance impact of three bad smells?	Dependency circle, poor use of abstract and shared database smell.	The smells have negative impact, depending on the smell, on response time, prevent independent scalability, database load.

their architectural variations (P2, P3, P5, P11, S1, S3, S6, S7, S8, S9, S13, S14, S15, S16, S17);

- Studies that model well-known microservice patterns and architectural styles to do a performance and scalability analysis (P10, P12);
- Studies that analyze the microservices design to identify and fix antipatterns that penalize performance (P1, P6, P9);
- Studies which are focused on the decomposition of a system in microservices and that propose microservice partitions that optimize performance and scalability (S4, S5, S12);
- A study that evaluates the stability of the test execution environment for microservice systems (P4);
- Studies that define generic assessment models for the microservice architecture (P7, P8, S2, S10, S11).

As can be observed, there is more emphasis on studies of particular systems or system architecture and also a few studies (3) on the decomposition process. Only 5 studies propose generic assessment models and one of these is actually more like a design method (S10). Therefore, only 4 studies have an analysis perspective. On the other hand, only 2 study well-known microservice patterns and architectural styles, but they do not do an extensive coverage of all patterns, [23] selected 7 patterns, and [34] focused on API patterns only.

Table 16
Goals, variations and results of studies P7-P12

Study	Goal	Variations	Results
P7	How can a model for performance of microservices be defined?	Horizontal and vertical scaling.	A novel methodology for model learning and verification of microservice-based systems under different deployment alternatives.
P8	How can a model for the occurrence of performance violations of microservices systems be defined?	Horizontal and vertical scaling.	A novel modeling approach for the analysis of transient performance behavior of microservices.
P9	How can a refactoring method for the improvement of performance based on the observation of the operation of a real system be defined?	Container-base (variations in terms of intra and inter container microservice interference).	Refactorings positively impact performance.
P10	What is the impact on performance of a set of microservices design patterns?	Anti-corruption layer, backends for frontends, command and query responsibility segregation, gateway aggregation, gateway offloading, pipe and filters, static content hosting.	Positive and negative impacts, depending on the pattern application. Also identify requirements for resources, like the need for parallelism, and for computation, like optimize operations.
P11	How to detect unnecessary data transfer that causes significant performance overhead in Java-based microservice applications?	Data transfers between microservices	Microservices enjoy up to 4.59× speedup in average latency when unnecessary data transfer on problematic micorservices are detected and optimized.
P12	What methods can be used to experimentally study and evaluate possible combinations of microservices API patterns to improve performance and scalability?	Request bundle, rate limit, load balancing.	Propose regression models that could be used for guidance. Load Balancing pattern has a clear negative impact on performance.

Therefore, we can conclude that more studies are needed on these two aspects.

Conversely, if the goals are diverse, then the achieved results are also different and very difficult to compare. We could even identify apparently contradictory results, which reflect the fact that most of the studies do not use the other studies as related work, and the conditions of experiment varies widely between studies. In some sense, there is a lack of a research corpus that researchers can build on, starting with a rich benchmark, containing the relevant architectural variations, that is used by the different studies, such that their results can be compared. As an illustration, performance evaluation when comparing monolithic and microservice implementations (S3, S6, S8, S16) shows that performance and scalability differences are often due to changes in workload dimensions. However, differences in architectural design in various studies also influence outcomes, making generalization difficult. A similar situation occurs with the proposals for performance and scalability quality assessment models, where each study proposes its own, but there is a lack of comparison between them (P7, P8, S2, S11).

Finally, in terms of variations in software architecture, we can observe that they occur mainly in aspects of deployment, such as horizontal vs. vertical scaling and in CPU, memory, etc., which is already a consequence of similar research done in cloud systems [52]. Only a few studies have reported, explicitly, on microservice architecture patterns and styles (P10, P12).

4.4. Recommendations for Future Research

In the previous subsection, we have found that the existing literature on performance and scalability assessment is focused on specific practical domains and lacks a holistic approach for performance and scalability. Therefore, we suggest that the body of knowledge on microservice architecture, performance, and scalability could benefit from additional research in these areas:

- Assessment models and methods of performance and scalability qualities for the microservice architecture, such that the obtained results can be generalized;
- Comparative studies of microservice architectural patterns and styles, such that in the architecture design of microservice, systems performance and scalability can be included in the trade-off behind architectural decisions;
- Scalability studies, because we observed that these studies present higher variability of methods, metrics and models;
- Assessment benchmarks that could be used across different studies, because we observed that a couple of reference systems are already being used (Table 10), like TrainTicket and SockShop, but they do not allow to compare many architectural variations (Tables 7, 8 and 9).

From the identified topics, we consider that the definition of a benchmark is core to have a clear scientific progress on the research on the qualities of performance and scalability of the microservice architecture. For instance, we

Table 17
Goals, variations and results of studies S1-S8

Study	Goal	Variations	Results
S1	What is the suitability of the CPU utilization for scaling microservices which are consuming messages from a message queue?	Autoscaling driven by queue sizes.	CPU utilization is a suitable metric for scaling all classes of microservices if they exhibit constant characteristics. Thresholds based on message queue metrics are much more resilient to changes in the microservice characteristics.
S2	How to define a quantitative approach for the performance assessment of microservice deployment alternatives?	Bare metal and virtualized environments. Horizontal and vertical scaling.	A new quantitative approach and framework for the assessment of microservice architecture configuration alternatives. Performance decreases with increasing workload. Horizontal scaling looks like having better results on virtualized environments.
S3	What is the performance considering the relationships between different variables of an application that runs in a monolithic structure compared to one of the microservices?	Monolith and microservices.	Average latency per request and throughput is lower and higher, respectively, on the microservices variation of the application.
S4	What is the impact of service decomposition on the performance?	Choreography, orchestration and use of cache.	Decomposed microservices show performance degradation under a lower number of concurrent users. Under higher number of concurrent users the performance can degrade or improve depending on workload characteristics of the individual microservice.
S5	How can a monolith be partitioned such that the chosen partition optimizes performance?	Monolith partitions.	The chosen partition has 11% less latency and 120% more throughput than the baseline.
S6	What are the performance and scalability differences of monolithic and microservice architectures on a reference web application?	Azure Spring Cloud and Azure App Service. Horizontal and vertical scaling.	On a single machine, a monolith performs better than its microservice-based counterpart. Scaling up the microservice-based application performed well and was more cost-efficient than horizontal scaling. Java and C# .NET have different advantages and drawbacks, there is no winner technology.
S7	What are the trade-offs between the dominant scaling techniques, including horizontal scaling, vertical scaling, and brownout in terms of execution cost and response time?	Horizontal, vertical and brownout scaling.	Vertical scaling converges faster to a stable connection time than horizontal and brownout scaling. Brownout approach initially achieves the best connection time. Horizontal scaling can achieve better connections than brownout since more resources are provided regarding the number of nodes.
S8	How performance and computing resource waste varies between a single monolithic codebase deployment and containerized microservice-based deployments?	Monolith and microservices; horizontal scaling.	The microservice architecture handled load increases much better in comparison. When response time did increase, the microservice-based architecture quickly added units of capacity in the form of more replicas of the service under the most load.

have observed that in the studies that analyze architectural variations, in addition to deployment variations, the test target is not a system (Table 14), which may reflect the limitations of existing reference systems.

Following [49, 50] the benchmark should provide a rich set of microservices that vary in terms of its business logic complexity, from read-only microservices to write microservices and computation intensive ones. The benchmark should allow two types of microservice coordination, orchestration and choreography. Upon this core, the benchmark should allow, by configuration, the experimentation of different architectural variations. In addition, the benchmark should support the definition of different metrics and operational profiles.

The implications to professionals working on large industrial projects concern the selection of methodologies, tools, and metrics for performance and scalability. The application of this methodology can be used to help uncover problems and guide architectural decisions to correct performance and scalability problems found in development and operations. Ideally, these methods must be integrated into the DevOps pipelines of the project.

Table 18
Goals, variations and results of studies S9-S17

Study	Goal	Variations	Results
S9	Does performance and scalability of freelance marketplace system varies for different inter-service communication technologies, in particular REST and GraphQL technologies?	Request aggregation and schema stitching (API composition); REST and GraphQL (API Gateway).	REST compatible gateway performs significantly better than the GraphQL gateway especially when aggregating data from multiple services.
S10	Is there an approach to systematically analyse which dimensions and metrics are important for scalability-aware granularity adaptation decisions for microservice-based applications?	Not applicable.	It argues that a catalog of microservice-specific scalability dimensions and metrics is needed to inform microservices adoption with scalability in mind.
S11	How to define a benchmarking method to conduct empirical scalability evaluations of cloud-native applications, frameworks, anddeployment options?	Asynchronous (Kafka Streams and Flink); horizontal scaling; cloud deployment platforms (Google Cloud Platform, Oracle Cloud Infrastructure, private cloud).	The benchmarking method consists of scalability metrics, measurement methods, and an architecture for a scalability benchmarking tool, particularly suited for cloud-native applications.
S12	Do actor-driven decomposition approaches result into more scalable microservices and which deployment architecture alternatives increase or decrease performance and scalability?	Horizontal scaling; resources (CPU and RAM); CQRS, Replicated Databases and Sharding.	Fine-grained and actor-driven decomposition approaches can be effectively used to further decompose microservices and achieve scalability improvements by taking proper architectural choices to better fit the target operational setting.
S13	How does load-balancing improve the balance of coordinated requests of a fully-distributed coordination layer of a microservice-based architecture to improve scalability and user-perceived performance?	Assignment to virtual machines (deployment); horizontal scaling; load balancing side.	Load balancing is able to effectively reduce the loss of user-perceived performance when the system undergoes high demand rates; the result is not the same with very low to medium loads (50–500 concurrent users).
S14	How performance and scalability of microservices-based architectures compare for Spring Cloud and Kubernetes platforms?	Infrastructure services (API Gateway, Load Balancer and Service Discover); horizontal scaling.	Frameworks and platforms for migration to a microservice architecture may have some impact on performance and scalability.
S15	How much variation is found in the scalability of state-of-the-art stream processing frameworks in different execution environments and regarding different scalability dimensions?	Horizontal and vertical scaling; container based on virtualized environment (public and private clouds); Asynchronous messaging.	Linear scalability is achieved generally as long as sufficient cloud resources are provisioned. The rate at which resources have to be added to cope with increasing load varies significantly.
S16	How performance and scalability of a monolith and variations of its microservice architecture implementation compare?	Horizontal scaling.	Microservice architecture improves scalability; significant performance degradation on cloud environments compared to local deployment.
S17	How performance of two communication methods (pool-based vs push-based) affects the tenant waiting time on a microservice based system?	Horizontal and vertical scaling; Asynchronous messaging (pool-based vs push-based).	Replication of components can positively affect system performance regarding average waiting time per tenant. Push-based presents a better average waiting time for all the scenarios, although not very significative.

5. Threats of validity

In this section, we discuss four potential threats of validity based on a standard checklist developed by Wohlin [53]. We discuss internal validity, external validity, construct validity, and conclusion validity.

Internal validity. One important threat to validity in this systematic literature review is that the research of scalability and performance assessment in the context of microservices architecture variation is in its early phases, and further data collection is necessary to draw definitive conclusions. Therefore, results might be affected by dimensions that are not included in the analyzed data. We have, however, broadened the scope of the studies selection to studies that address different architectural aspects of microservice architectures even without presenting variations on architectural elements.

Construct validity. Another type of threat to validity is that incomplete or inadequately designed search strategies might miss relevant studies. Also, search strings may be poorly defined and relevant studies may be missed. To address this last issue, we have applied the QGS-based systematic search approach ([24]) that evaluates the quasi-sensitivity of the automated search. To mitigate this threat, we used a multi-step approach to ensure the quality of the automated search.

External validity. Concerning the applicability of the results of our study in a more general context, we have focused on the approaches described in the selected studies. Our result consists of a synthesis of the findings of these primary studies. For the selected timeframe, we are confident that we have encompassed a broad range of architectural design elements and architecture quality assessment domains, along with their various methods. Additional or emerging approaches can also be evaluated using the proposed framework of analysis.

Conclusion validity. The nature of the data from the literature review is not conducive to the application of statistical analysis to select the performance and scalability dimensions that are the most appropriate to be integrated into the framework of analysis. To mitigate this threat, we used performance engineer expert advise to support the selection of the performance and scalability dimensions.

6. Related Work

Several systematic literature reviews and surveys have addressed the relation between software architecture and performance [54, 55, 56, 57, 58].

Balsamo et al. [54] identify that traditional software development methods do not address performance in the early stages of software development. Therefore, they review research in the field of model-based performance prediction to analyze the feasibility of addressing performance early, concluding that it can be applied in the architectural phase. Their focus is on the development process and not the interrelationships between performance and software architecture.

Also focusing on software development activities, [55] analyzes several empirical studies on the evaluation of software architectures. They conclude on the importance of empirical studies, but their focus is not on performance and do not analyze the impact that architectural variations have on performance.

Aleti et al. [56] do a systematic review of the research literature on architecture optimization, defining a classification taxonomy. Performance is identified as one of the most popular constraints addressed by analysis approaches. However, they do not do a detailed comparison of the different approaches, and the emphasis is more on the methods and do not address architectural variations. Additionally, the microservice architecture is not addressed.

The applicability of various performance prediction methods for the development of component-based systems is analyzed in [57]. They concluded that queuing networks are the most commonly used model. Although the architecture style analyzed is modular, there are significant differences from the microservice architecture and their variations.

A more recent study [58] on the architectural approaches to performance analysis also includes the microservice architecture. Their research questions are about the purpose of the analysis and automation of the tools. They conclude that there is a lack of available tools and benchmark datasets to support replication, cross-validation and comparison of studies, and a need for the adoption of modern ML/AI techniques. Although they discuss some architectural patterns, they do not provide a systematic framework or a detailed analysis.

The studies that consider the microservice architecture in particular are scarce, though more recent [59, 60, 61, 62, 63].

Vural et al. [59] present a systematic literature review of microservice architectures, describing new trends in service-oriented computing and cloud computing. However, they do not analyze the impact on performance and scalability nor the impact of architectural variations.

Waseem et al. [60] have conducted a systematic mapping study to classify the literature on microservices architecture in DevOps. Performance is one of the aspects addressed, where they have identified performance issues due to frequent communication. Since they cover more aspects than performance and scalability, they lack a detailed analysis of these quality attributes. Moreover, most of the research publications that we analyzed are after 2020, indicating the relevance of the subject and the need for new studies.

The systematic literature survey in [61] identified performance and scalability as two of the six quality attributes most relevant to microservice architecture, which is an excellent justification for our systematic study.

Almeida and Canedo [62] present a systematic literature review on the authentication and authorization in microservice architectures, their focus is not on scalability and performance.

We have conducted a literature survey reporting on the performance of monolithic systems with their microservice architecture implementations [5]. It analyzes the architectural variations of the studies, but only in the context of studies that compare monoliths with their refactored microservice system, and do not provide a holistic assessment framework.

A more recent systematic mapping study [63] analyzes systems used in research to test and monitor microservice-based systems, which is similar to our benchmark dimension. However, they focus on the research goals of the studies that use systems. Interestingly, they observe that only two studies address architecture.

Therefore, with the exception of our previous work, the related work consists mainly of papers on microservice architecture, or performance and scalability qualities of microservice architecture. To our knowledge, this is the first systematic review of the literature on architecture and performance scalability *and* scalability qualities of microservice architectures.

7. Conclusion

In this paper, we introduced an analysis framework to support a systematic literature review of performance and scalability qualities assessment of microservice architectures. The analysis framework consists of a microservice architecture reference model and a quality assessment framework.

We defined a multi-step literature review methodology that was used to select 29 studies after a systematic analysis of 801 initial studies. The search period was for papers published between 2014 and the beginning of 2024. We also performed a quality check to ensure that the selected papers comply with the defined quality checklist.

We performed a detailed analysis of the 29 studies using the two introduced frameworks: architecture and quality assessment. The analysis created a mapping between the framework dimensions and each analyzed paper.

We have answered two research questions. The **first research question** was addressed by analyzing how the selected studies cover different aspects of the architectural elements and concerns described in the microservice architecture reference model. The main contribution of this analysis was the observation that most studies focus on deployment and scaling alternatives for performance and scalability assessment, while variations in service orchestration elements are considered but rarely evaluated for their impact. In addition, although there are studies on communication styles and data management consistency issues in microservice architecture [64, 65], they are not integrated in the performance and scalability assessment. These integrations are crucial for the assessment of distributed transactions and scalability. In addition, the chosen codebases for benchmarking often lack justification for their comprehensive coverage of architectural elements.

The **second research question** was answered by addressing quantitative methodologies for the assessment of performance and scalability qualities in the literature. Each of the quality assessment domains defined in the assessment framework was analyzed. The main contribution of this analysis was the identification of areas for further research, as we have encountered possible methodological gaps in the reviewed literature.

In addition, we analyze the goals and results of each of the studies while also focusing on the variations explored. We have performed comparisons of studies and tried to identify the main gaps in the assessment of microservice scalability and performance qualities in the context of microservice architectures. Although most studies focus on specific systems or system architectures, only a few propose generic assessment models or evaluate well-known microservice patterns. In addition, the lack of a standardized research corpus and the varying experimental conditions make it difficult to compare the results between studies.

Finally, we have identified a set of open research trends to guide researchers in the field of microservice architecture quality assessment.

Acknowledgment

This work was partially supported by eSulabSolutions, Inc. and the Fundação para a Ciência e Tecnologia (FCT) through projects UIDB/50021/2020 (INESC-ID) and 2024.07494.IACDC (WELL).

References

[1] Charlene O'Hanlon. A conversation with werner vogels. *Queue*, 4(4):14–22, May 2006.

[2] Martin Fowler. Microservices. Web page: `http://martinfowler.com/articles/microservices.html`, 2014. Accessed: 2023-07-06.

[3] J. Thönes. Microservices. *IEEE Software*, 32(1):116–116, 2015.

[4] Diogo Faustino, Nuno Gonçalves, Manuel Portela, and António Rito Silva. Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation. *Performance Evaluation*, 164:102411, 2024.

[5] Helena Rodrigues, António Rito Silva, and Alberto Avritzer. Performance comparison of monolith and microservice architectures: An analysis of the state of the art. In *1st International Workshop on Quality in Software Architecture (QUALIFIER 2023), co-located with the 17th European Conference on Software Architecture (ECSA) 2023, Istanbul, Turkey, September 2023*. Springer, 2023.

[6] Andre B. Bondi. Characteristics of scalability and their impact on performance. In *Second International Workshop on Software and Performance, WOSP 2000, Ottawa, Canada, September 17-20, 2000*, pages 195–203. ACM, 2000.

[7] Elaine J. Weyuker and Alberto Avritzer. A metric for predicting the performance of an application under a growing workload. *IBM Syst. J.*, 41(1):45–54, 2002.

[8] Ming Yan, XiaoMeng Liang, ZhiHui Lu, Jie Wu, and Wei Zhang. Hansel: Adaptive horizontal scaling of microservices using bi-lstm. *Applied Soft Computing*, 105:107216, 2021.

[9] N. C. Mendonca, C. Box, C. Manolache, and L. Ryan. The monolith strikes back: Why istio migrated from microservices to a monolithic architecture. *IEEE Software*, 38(05):17–22, sep 2021.

[10] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University, UK and Lincoln University, NZ, 2007.

[11] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond, Second Edition*. SEI Series in Software Engineering. Addison-Wesley, Upper Saddle River, NJ, 2010.

[12] Chris Richardson. Developing transactional microservices using aggregates, event sourcing and cqrs. *InfoQ*, 2017.

[13] Chris Richardson. *Microservices Patterns*. Manning Publications Co., 2019.

[14] Mehmet Söylemez, Bedir Tekinerdogan, and Ayça Kolukısa Tarhan. Microservice reference architecture design: A multi-case study. *Software: Practice and Experience*, 54(1):58–84, 2024.

[15] Armando Fox and Eric A. Brewer. Harvest, yield, and scalable tolerant systems. In *Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, HOTOS '99, page 174, USA, 1999. IEEE Computer Society.

[16] Hector Garcia-Molina and Kenneth Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, SIGMOD '87, page 249–259, New York, NY, USA, 1987. Association for Computing Machinery.

[17] Alberto Avritzer, Vincenzo Ferme, Andrea Janes, Barbara Russo, André van Hoorn, Henning Schulz, Daniel Menasché, and Vilc Rufino. Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests. *Journal of Systems and Software*, 165:110564, 2020.

[18] Matteo Camilli and Barbara Russo. Modeling performance of microservices systems with growth theory. *Empirical Softw. Engg.*, 27(2), mar 2022.

[19] Matteo Camilli, Carmine Colarusso, Barbara Russo, and Eugenio Zimeo. Actor-driven decomposition of microservices through multilevel scalability assessment. *ACM Trans. Softw. Eng. Methodol.*, 32(5), jul 2023.

[20] Sara Osama Hassan, Rami Bahsoon, and Rajkumar Buyya. Systematic scalability analysis for microservices granularity adaptation design decisions. *Software: Practice and Experience*, 52:1378 – 1401, 2022.

[21] Matteo Camilli, Andrea Janes, and Barbara Russo. Automated test-based learning and verification of performance models for microservices systems. *J. Syst. Softw.*, 187(C), may 2022.

[22] Vittorio Cortellessa, Daniele Di Pompeo, Romina Eramo, and Michele Tucci. A model-driven approach for continuous performance engineering in microservice-based systems. *Journal of Systems and Software*, 183:111084, January 2022.

[23] Riccardo Pinciroli, Aldeida Aleti, and Catia Trubiani. Performance modeling and analysis of design patterns for microservice systems. In *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, pages 35–46, 2023.

[24] He Zhang, Muhammad Ali Babar, and Paolo Tell. Identifying relevant studies in software engineering. *Information and Software Technology*, 53(6):625–637, 2011.

[25] Raminta Pranckutė. Web of science (Wos) and scopus: The titans of bibliographic information in today's academic world. *Publications*, 9(1), 2021.

[26] Martijn Visser, Nees Jan van Eck, and Ludo Waltman. Large-scale comparison of bibliographic data sources: Scopus, web of science, dimensions, crossref, and microsoft academic. *Quantitative Science Studies*, 2(1):20–41, 2021.

[27] Catia Trubiani, Anne Koziolek, Vittorio Cortellessa, and Ralf Reussner. Guilt-based handling of software performance antipatterns in palladio architectural models. *Journal of Systems and Software*, 95:141–165, 2014.

[28] Marcelo Amaral, Jordà Polo, David Carrera, Iqbal Mohomed, Merve Unuvar, and Malgorzata Steinder. Performance evaluation of microservices architectures using containers. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, pages 27–34, 2015.

[29] Mario Villamizar, Oscar Garcés, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS lambda architectures. *Service Oriented Computing and Applications*, 11:233–247, 2017.

[30] Simon Eismann, Cor-Paul Bezemer, Weiyi Shang, Dušan Okanović, and André van Hoorn. Microservices: A performance tester's dream or nightmare? In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page 138–149, New York, NY, USA, 2020. Association for Computing Machinery.

[31] Devki Nandan Jha, Saurabh Garg, Prem Prakash Jayaraman, Rajkumar Buyya, Zheng Li, Graham Morgan, and Rajiv Ranjan. A study on the evaluation of hpc microservices in containerized environment. *Concurrency and Computation: Practice and Experience*, 33(7):e5323, 2021. e5323 cpe.5323.

[32] Lei Liu, Zhiying Tu, Xiang He, Xiaofei Xu, and Zhongjie Wang. An empirical study on underlying correlations between runtime performance deficiencies and "bad smells" of microservice systems. In *2021 IEEE International Conference on Web Services (ICWS)*, pages 751–757, 2021.

[33] Syed Salauddin Mohammad Tariq, Lance Menard, Pengfei Su, and Probir Roy. Microprof: Code-level attribution of unnecessary data transfer in microservice applications. *ACM Trans. Archit. Code Optim.*, 20(4), dec 2023.

[34] Amine El Malki and Uwe Zdun. Combining api patterns in microservice architectures: Performance and reliability analysis. In *2023 IEEE International Conference on Web Services (ICWS)*, pages 246–257, 2023.

[35] Manuel Gotin, Felix Lösch, Robert Heinrich, and Ralf Reussner. Investigating performance metrics for scaling microservices in cloudiot-environments. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, page 157–167, New York, NY, USA, 2018. Association for Computing Machinery.

[36] Freddy Tapia, Miguel Angel Mora, Walter Fuertes, Hernán Aules, Edwin Flores, and Theofilos Toulkeridis. From monolithic systems to microservices: A comparative study of performance. *Applied Sciences*, 10(17), 2020.

[37] Malith Jayasinghe, Jayathma Chathurangani, Gayal Kuruppu, Pasindu Tennage, and Srinath Perera. An analysis of throughput and latency behaviours under microservice decomposition. In Maria Bielikova, Tommi Mikkonen, and Cesare Pautasso, editors, *Web Engineering*, pages 53–69, Cham, 2020. Springer International Publishing.

[38] Vikram Nitin, Shubhi Asthana, Baishakhi Ray, and Rahul Krishna. Cargo: Ai-guided dependency analysis for migrating monolithic applications to microservices architecture. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery.

[39] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybyłek. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, 10:20357–20374, 2022.

[40] Minxian Xu, Chenghao Song, Shashikant Ilager, Sukhpal Singh Gill, Juanjuan Zhao, Kejiang Ye, and Chengzhong Xu. Coscal: Multifaceted scaling of microservices with reinforcement learning. *IEEE Transactions on Network and Service Management*, 19(4):3995–4009, 2022.

[41] Aos Mulahuwaish, Shane Korbel, and Basheer Qolomany. Improving datacenter utilization through containerized service-based architecture. *Journal of Cloud Computing*, 11(1):44, 2022.

[42] Naman Vohra and Ida Bagus Kerthyayana Manuaba. Implementation of rest api vs graphql in microservice architecture. In *2022 International Conference on Information Management and Technology (ICIMTech)*, pages 45–50, 2022.

[43] Sören Henning and Wilhelm Hasselbring. A configurable method for benchmarking scalability of cloud-native applications. *Empirical Software Engineering*, 27(6):143, 2022.

[44] Gianluca Filippone, Claudio Pompilio, Marco Autili, and Massimo Tivoli. An architectural style for scalable choreography-based microservice-oriented distributed systems. *Computing*, 105(9):1933–1956, dec 2022.

[45] Yu-Te Wang, Shang-Pin Ma, Yue-Jun Lai, and Yan-Cih Liang. Qualitative and quantitative comparison of spring cloud and kubernetes in migrating from a monolithic to a microservice architecture. *Service Oriented Computing and Applications*, 17(3):149–159, 2023.

[46] Sören Henning and Wilhelm Hasselbring. Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud. *Journal of Systems and Software*, 208:111879, February 2024.

[47] Cesar Batista, Felipe Morais, Everton Cavalcante, Thais Batista, Bruno Proença, and William Breno Rodrigues Cavalcante. Managing asynchronous workloads in a multi-tenant microservice enterprise environment. *Software: Practice and Experience*, 54(2):334–359, 2024.

[48] Neal Ford, Ford Richards, Pramod Sadalage, and Zhamak Dehghani. *Software Architecture: The Hard Parts*. O'Reilly Media, Inc., 2021.

[49] S.E. Sim, S. Easterbrook, and R.C. Holt. Using benchmarking to advance research: a challenge to software engineering. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 74–83, 2003.

[50] Jóakim v. Kistowski, Jeremy A. Arnold, Karl Huppler, Klaus-Dieter Lange, John L. Henning, and Paul Cao. How to build a benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, page 333–336, New York, NY, USA, 2015. Association for Computing Machinery.

[51] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 3–18, New York, NY, USA, 2019. Association for Computing Machinery.

[52] Sebastian Lehrig, Hendrik Eikerling, and Steffen Becker. Scalability, elasticity, and efficiency in cloud computing: a systematic literature review of definitions and metrics. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, QoSA '15, page 83–92, New York, NY, USA, 2015. Association for Computing Machinery.

[53] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery.

[54] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Trans. Softw. Eng.*, 30(5):295–310, May 2004.

[55] Davide Falessi, Muhammad Ali Babar, Giovanni Cantone, and Philippe Kruchten. Applying empirical software engineering to software architecture: challenges and lessons learned. *Empirical Softw. Engg.*, 15(3):250–276, June 2010.

[56] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2013.

[57] Matthias Becker, Markus Luckey, and Steffen Becker. Model-driven performance engineering of self-adaptive systems: a survey. In *Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures*, QoSA '12, page 117–122, New York, NY, USA, 2012. Association for Computing Machinery.

[58] Yutong Zhao, Lu Xiao, Chenhao Wei, Rick Kazman, and Ye Yang. A systematic mapping study on architectural approaches to software

performance analysis. *arXiv*, 2410.17372, 2024.

[59] Hulya Vural, Murat Koyuncu, and Sinem Guney. A systematic literature review on microservices. In Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra, Giuseppe Borruso, Carmelo M. Torre, Ana Maria A.C. Rocha, David Taniar, Bernady O. Apduhan, Elena Stankova, and Alfredo Cuzzocrea, editors, *Computational Science and Its Applications – ICCSA 2017*, pages 203–217, Cham, 2017. Springer International Publishing.

[60] Muhammad Waseem, Peng Liang, and Mojtaba Shahin. A systematic mapping study on microservices architecture in devops. *Journal of Systems and Software*, 170:110798, 2020.

[61] Shanshan Li, He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen, and Muhammad Ali Babar. Understanding and addressing quality attributes of microservices architecture: A systematic literature review. *Information and Software Technology*, 131:106449, 2021.

[62] Murilo Góes de Almeida and Edna Dias Canedo. Authentication and authorization in microservices architecture: A systematic literature review. *Applied Sciences*, 12(6), 2022.

[63] Stefan Fischer, Pirmin Urbanke, Rudolf Ramler, Monika Steidl, and Michael Felderer. An overview of microservice-based systems used for evaluation in testing and monitoring: A systematic mapping study. In *2024 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 182–192, 2024.

[64] Işıl Karabey Aksakalli, Turgay Çelik, Ahmet Burak Can, and Bedir Tekinerdoğan. Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, 180:111014, 2021.

[65] Rodrigo Laigner, Yongluan Zhou, Marcos Antonio Vaz Salles, Yijian Liu, and Marcos Kalinowski. Data management in microservices: state of the practice, challenges, and research directions. *Proc. VLDB Endow.*, 14(13):3348–3361, September 2021.