Scientific Research Publishing

Open Journal of Applied Sciences, 2024, 14, 2519-2543
https://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

Current Trends in the Management of Distributed Transactions in Micro-Services Architectures: A Systematic Literature Review

Samuel Lungu, Mayumbo Nyirenda

Department of Computer Studies, University of Zambia, Lusaka, Zambia
Email: samuel.lungu@unza.zm, mayumbo@gmail.com

How to cite this paper: Lungu, S. and Nyirenda, M. (2024) Current Trends in the Management of Distributed Transactions in Micro-Services Architectures: A Systematic Literature Review. *Open Journal of Applied Sciences*, **14**, 2519-2543.
https://doi.org/10.4236/ojapps.2024.149167

Received: August 13, 2024
Accepted: September 22, 2024
Published: September 25, 2024

Abstract

In the evolving landscape of software engineering, Microservice Architecture (MSA) has emerged as a transformative approach, facilitating enhanced scalability, agility, and independent service deployment. This systematic literature review (SLR) explores the current state of distributed transaction management within MSA, focusing on the unique challenges, strategies, and technologies utilized in this domain. By synthesizing findings from 16 primary studies selected based on rigorous criteria, the review identifies key trends and best practices for maintaining data consistency and integrity across microservices. This SLR provides a comprehensive understanding of the complexities associated with distributed transactions in MSA, offering actionable insights and potential research directions for software architects, developers, and researchers.

Keywords

Microservice Architecture, Distributed Transactions, Two-Phase Commit (2PC)

1. Introduction

In today's fast-paced business landscape, organizations seek ways to enhance their software systems, streamline development processes, and respond swiftly to market demands. One groundbreaking approach that reshapes application design, development, and deployment is the Microservice Architecture (MSA). At its core, MSA breaks down complex applications into smaller, self-contained services [1]. Each service operates independently, communicating seamlessly with others using lightweight protocols like HTTP and REST [2]. Imagine your business

application as a collection of interconnected building blocks, each responsible for a specific function: handling payments, managing user profiles, or processing orders.

Microservice Architecture (MSA) provides significant advantages over traditional monolithic architectures. These advantages include enhanced scalability, agility, and the ability to deploy services independently [3]. Additionally, MSA encourages the adoption of a design pattern called "atabase-per-service?" allowing each microservice to manage its database system. This pattern permits each microservice to utilize the most suitable database technology (SQL, NoSQL, etc.) based on its specific requirements. One of the benefits of the "database-per-service" design pattern is data isolation where changes or issues affecting one microservice's database won't impact others.

To simplify data management, applications utilize the concept of a transaction, where a series of read-and-write operations are grouped into a single unit. Conceptually, all the reads and writes in a transaction are executed as one operation: either the entire transaction succeeds (commit) or it fails (abort, rollback) [4]. A Database Management System (DBMS) is said to be transactional if it supports ACID (Atomicity, Consistency, Isolation, Durability) properties. ACID properties guarantee that database transactions are processed reliably [5]. In a monolithic application managing transactions is more straightforward because the application only interacts with one homogeneous database. However, the "database-per-service" design pattern, brings in the dynamics of distributed transactions, where a single transaction may span two or more microservices. The foremost challenge of coordinating distributed transactions is ensuring data consistency and integrity. As MSA continues to evolve and proliferate, understanding the current trends in the management of distributed transactions becomes paramount for architects, developers, and researchers alike.

This article explores the current state of the art and emerging trends in managing distributed transactions within microservices architectures (MSA). To achieve this, we will conduct a systematic literature review (SLR) [6]. The SLR aims to provide a comprehensive overview of the challenges, best practices, and research directions related to distributed transaction management in MSA. Through critical analysis of existing research studies, we will identify key patterns, strategies, and technologies used to address the unique challenges posed by distributed transactions in MSA environments.

The collection of these research studies will cover a period of 10 years starting from 2014 when MSA was officially coined [7] to the present year, 2024. We specifically focus on research conducted during this period to capture the most recent developments in this dynamically evolving field. The scope of this review encompasses studies published in peer-reviewed journals, conference proceedings, and other relevant academic sources. By shedding light on the current landscape of distributed transaction management in microservices architectures, we aim to equip practitioners and researchers with insight into the latest advancements, best practices, and areas warranting further investigation.

Our research findings seek to enhance the ongoing discourse on designing, implementing, and managing microservices-based systems in the context of distributed computing. Additionally, this review study serves as a crucial component in fulfilling the primary author's Master's degree requirements.

The article is organized as follows. Section 3 delves into related works, Section 4 outlines the methodology, Section 5 details the results and examines potential threats to validity, Section 6 engages in discussion, and Section 7 provides the conclusion.

2. Related Works

Since the inception of microservices, a range of secondary studies have been undertaken to dissect the multifaceted nature of microservices [8]-[12]. In this context, a secondary study refers to the analysis, synthesis, and interpretation of existing research rather than the collection of new data. Examples of secondary studies, therefore, are mapping studies and literature reviews. Subsequent subsections highlight some of the conducted secondary studies organized by topic.

2.1. Security

A study by Berardi *et al.* [12] focused on security in microservices, the researchers conducted a systematic literature review by gathering 290 relevant publications and analyzed responses to 20 research questions. Consequently, the study made some recommendations such as the need for adopting "security-by-design" principles throughout the lifecycle of a microservice application. Further, it emphasized the need to have guidelines on how to reliably implement these "security-by-design" principles and well-defined techniques to assist developers in migrating legacy systems to microservice architectures.

2.2. Deployment and Communication Strategies

In a study by Aksakalli [10], the researchers aimed to identify and describe the deployment approaches, and the communication platforms for microservices. They identified seven different types of communication patterns and three deployment approaches with Publish/subscribe communication (one-to-many interactions) being the most frequently used method among the identified communication patterns. Furthermore, Service Instance per Container Pattern was the most preferred deployment approach in the selected studies.

2.3. Data Management

One notable study by Laigner *et al.* [11] delves into the complexities of data management within microservices. The authors engaged with over 120 experienced practitioners and researchers, reviewed existing literature, and evaluated open-source microservice applications. Their findings highlighted a significant shortfall in the capabilities of current database systems to effectively manage data within microservices architecture, forcing developers to create ad-hoc solutions to meet

their needs.

Our review focuses specifically on the challenges associated with the management of distributed transactions and proposed solutions to overcome those challenges.

3. Methodology

This article aims to understand the current state of the art and trends in the management of distributed transactions in MSA. The study focuses particularly on the challenges and respective solutions in managing distributed transactions within MSA. To accomplish the study's aim, we conducted a systematic literature review by following guidelines set forth by Kitchenham and Charters [13]. Kitchenham's guide has proven highly effective for conducting systematic reviews, leading to its adoption in numerous studies [14]-[18]. The basic activities of the SLR as per Kitchenham's guidelines are illustrated in **Figure 1**.

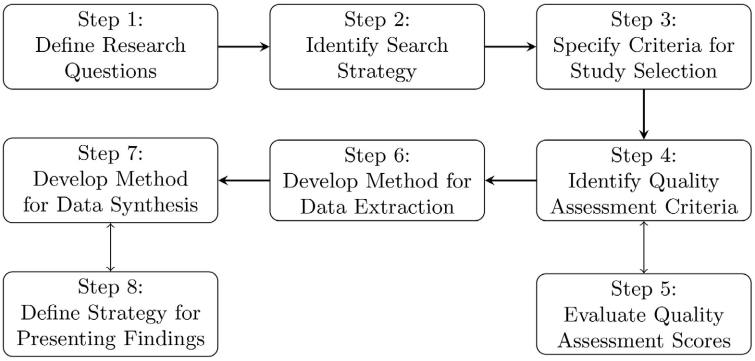

Figure 1. Activities under the review protocol.

3.1. Step 1: Define Research Questions

The objective of this study was to examine the current state of managing distributed transactions within Microservices Architecture (MSA). It aimed to analyze the challenges associated with handling distributed transactions in MSA and to explore the solutions addressing the identified challenges. In this context the research questions were as follows:

RQ1: What are the challenges associated with managing distributed transactions within MSA?

RQ2: What specific solutions address the identified challenges?

3.2. Step 2: Identify Search Strategy

In a Systematic Literature Review (SLR), a search strategy is a comprehensive and systematic process that outlines plans to find and retrieve all relevant scholarly literature of interest to the study. Designing and executing an effective search strategy in an SLR is a very critical step of the research, failure to which, the process can be more time-consuming and error-prone. Several methods have been suggested for conducting a search strategy. A manual literature search involves

manually searching specific venues, article by article looking for studies of interest, which is more labor-intensive and time-consuming. Database search (DBS), is another technique that involves coming up with a specific search string which is then used to search for primary studies of interest in different database engines [19]. The snowballing search (SB) strategy involves starting with a core set of papers and then examining their references (backward snowballing) and citations (forward snowballing) to discover additional papers [20]. This process is repeated, effectively snowballing? into a larger set of pertinent literature. Wohlin *et al.* [20] compared snowballing to Database search, and concluded that the two strategies are effectively equal in performance. An even better search strategy is a hybrid search strategy as proposed by Wohlin *et al.* [21]. They defined a hybrid search strategy as "a pre-planned integration of at least two systematic approaches to searching for articles for a Systematic Literature Study". In this study [21] a DBS was systematically combined with SB, both backward and forward snowballing. After evaluation, the hybrid search strategy emerged superior by identifying 30% more primary studies than DBS alone. Therefore, in this SLR study, we used a hybrid search strategy [21] to gather primary studies of interest. This hybrid strategy involves first gathering a start set using a database search, then performing backward snowballing on this start set before finally performing forward snowballing. The major steps in this strategy are as indicated in the following sub-steps:

3.2.1. Step I: Identify Start Set for Snowballing

The initial step in the hybrid search strategy is to identify a start set of articles, which is then used for snowballing in subsequent steps [20]. To create this start set, a search string is constructed based on the research questions and period of interest for the study. To avoid publisher bias, a publisher-agnostic indexing service, Google Scholar (GS) was used. In a study considering the suitability of GS as a source of scientific information Halevi *et al.* [22] argued that lack of indexing policy and transparency, duplications, lack of advanced search options, and inability to apply quality control to its indexing capabilities prevent GS from becoming a sole source of scholarly retrieval. In contrast, Cusker *et al.* [23] compared GS with Compendex and concluded that GS can be used as a primary literature search engine for Engineering owing to its free service. To avoid the inherent lack of quality of GS's published articles as argued [22], we performed a quality assessment on the sifted articles as indicated in subsection 4.4. Moreover, the initial start set of studies retrieved from GS was further complemented by backward and forward snowballing.

The primary studies were searched in their respective Titles, Abstracts, and Keywords sections. The search was confined to a decade-long period, starting from 2014 and extending to the present year, 2024. The year 2014 was chosen as a starting point because it is believed to be when the term "microservice(s)" was coined [7]. The search string for the start set was as indicated in **Table 1** and the options to include patents and citations were left unchecked.

Table 1. Search strings.

Search String
("distributed transactions") AND ("microservices" OR "microservice architecture")

The GS search was conducted on the 17th of April 2024, and it returned a compilation of 764 (Figure 2) scholarly articles. Thereafter, the returned articles were individually added to the first author's library, and the whole library was exported to an external file in both CSV and BibTeX format for further analysis. In compliance with the snowballing procedure as guided by Wohlin *et al.* [20] and as depicted in Figure 3, this initial compilation was then evaluated according to the inclusion/exclusion criteria as per Table 2. Each inclusion/exclusion criteria has been given a specific code along with a corresponding description.

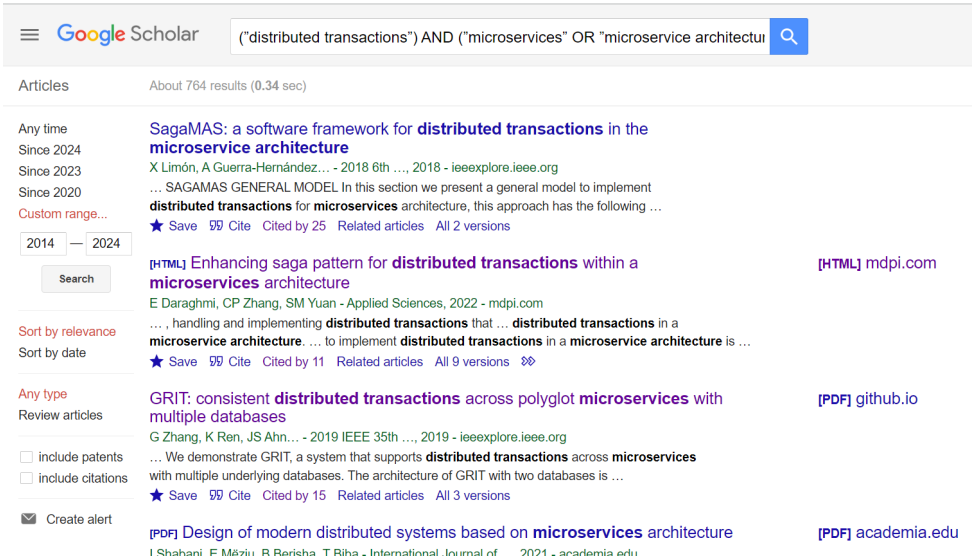

Figure 2. Initial search results.

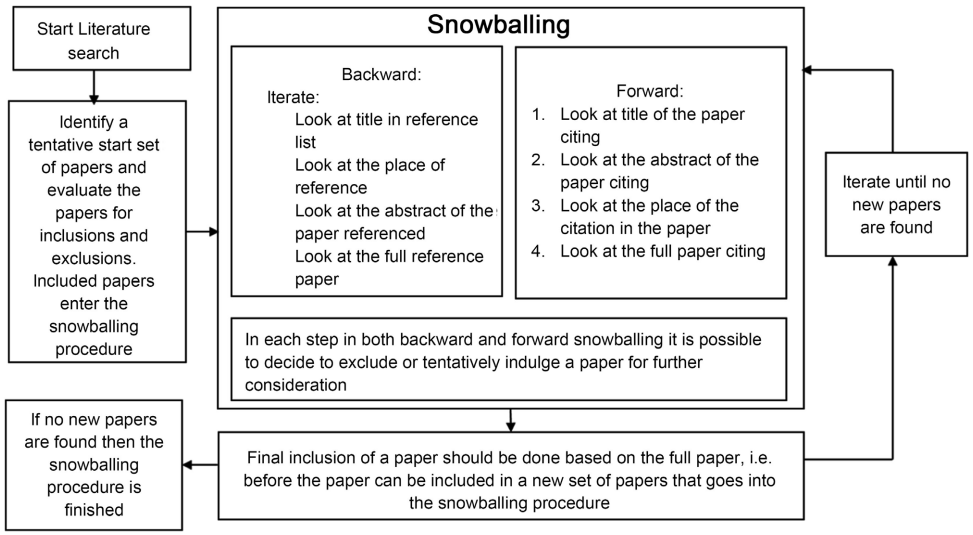

Figure 3. Snowballing procedure.

Table 2. Inclusion/Exclusion criteria.

Criteria	Code	Description
Inclusion (IC)	IC1	The study is written in the English language
	IC2	Title or abstract/keywords include key terms
	IC3	The abstract of the study indicates that the study is about distributed transactions in microservices
Exclusion (EC)	EC1	Studies not written in the English language
	EC2	Studies whose titles do not discuss microservices.
	EC3	Studies that are identified as reviews, literature surveys, secondary or tertiary studies
	EC4	Studies that are not peer-reviewed or published in reputable academic journals or conferences
	EC5	Studies whose titles focus on the application of microservices but do not address distributed transactions in microservices
	EC6	Studies whose abstracts do not discuss or propose an approach to distributed transactions in microservices

For an article to be considered for inclusion in our study, it was required to meet all the specified inclusion criteria while simultaneously not meeting any of the exclusion criteria. The forthcoming sections are focused on the exclusion criteria to provide a detailed examination of each criterion. The inclusion criteria are not discussed in detail because they are assumed to be universally applied across all studies meeting the research objectives.

EC1—Studies not written in the English language. This exclusion criteria is a complement of inclusion criteria IC1. GS makes specifying the languages of the primary studies easy by providing a languages sub-tab under the settings main tab. In compliance with this EC, English was ticked as a language of choice and the returned articles were reduced to 703.

EC2—Studies whose titles do not discuss microservices. Upon analyzing and screening the articles initially returned by the GS, we noticed that a good number of articles returned were false positives and irrelevant to our study. This echoed a similar experience shared by Kitchenham "...initial electronic searches results in large numbers of totally irrelevant papers" [13]. EC2 assisted in removing a total of 128 irrelevant articles.

EC3—Studies that are identified as reviews, literature surveys, secondary or tertiary studies. A secondary study is defined as a study that reviews all the primary studies related to a specific research question to integrate/synthesize collaborating evidence [13]. These secondary studies can be classified as mapping studies, literature reviews, systematic literature reviews, or general reviews. On the other hand, a tertiary study is a study of secondary studies. A total of 19 articles were excluded based on this exclusion criteria.

EC4—Studies that are not peer-reviewed or published in reputable academic

journals or conferences. This EC helped to drop the biggest number of articles amounting to 434. The criteria aims to prioritize studies that have been subjected to rigorous peer review and have been published in reputable academic outlets. This helps ensure the reliability, validity, and credibility of the sources included in the review and contributes to the overall quality of the research findings and conclusions.

EC5—Studies whose titles focus on the application of microservices but do not address distributed transactions in microservices. This exclusion criteria aims to exclude studies that discuss only the application of microservices without reflecting on distributed transactions. This helps to maintain the specificity and relevance of the literature review to the research objectives. A total of 91 articles were dropped based on this EC.

EC6—Studies whose abstracts do not discuss or propose an approach to distributed transactions in microservices. By applying EC6, the literature review aims to narrow down the pool of potential sources to those that are directly relevant to the topic of distributed transactions in microservices. It is interesting to mention that a total of 13 articles [24]-[36] were excluded based on this criteria. Some articles on this list would have made it on the final list if we considered EC5 alone which largely assesses the study's title. A case in point is article [26] entitled "The saga pattern in a reactive microservices environment", a critical analysis of the abstract in line with EC5 showed that this article is a review of four Java-based SAGA application frameworks for microservices. EC5 therefore, helps ensure that this SLR focuses on studies that provide insights and solutions that specifically address the research questions and objectives.

After completing the exclusion criteria exercise, we remained with a total of 16 articles. However, we could not access two [37] [38] of the studies due to access restrictions on SpringerLink. This further reduced the initial start set to 14 articles, which were thereafter used in the subsequent backward and forward snowballing steps.

3.2.2. Step II: Backward Snowballing

Backward Snowballing (BS) involves going through each publication's reference list to find additional publications that can be considered in the SLR [20]. The process of reference checking was done under the strict guide of inclusion/exclusion criteria. BS facilitated the identification of 2 more articles, thus bringing the total number of articles to 16.

3.2.3. Step III: Forward Snowballing

According to Wohlin *et al.* [20] Forward Snowballing (FS) is the process of identifying new papers based on those papers citing the paper being examined. The citations were easier to collect in Google Scholar since it was our primary source of primary studies. FS process could only identify articles that were already on the list—duplicates, which could not be included. One key aspect of the backward and forward snowballing procedure as guided by Wohlin *et al.* [20] is that they are

iterative processes. Iteration played a key role during the whole search process to ensure thoroughness in the methodology. The total number of articles considered for further analysis was still 16 and is listed in Table 3. Each article was given an article ID, the year published, the title, and the venue.

Table 3. List of selected articles.

Id	Year	Title	Venue
P1	2015	Scalable distributed transactions across heterogeneous stores	Conference
P2	2016	Transactions for Distributed Actors in the Cloud	Technical Report
P3	2017	Transaction management across data stores	Journal
P4	2018	SagaMAS: a software framework for distributed transactions in the microservice architecture	Conference
P5	2019	GRIT: Consistent Distributed Transactions Across Polyglot Microservices with Multiple Databases	Conference
P6	2020	Fault-tolerant central saga orchestrator in RESTful architecture	Conference
P7	2020	2PC*: a distributed transaction concurrency control protocol of multi-microservice based on cloud computing platform	Journal
P8	2020	Enhancing Performance of Distributed Transactions in Microservices via Buffered Serialization	Journal
P9	2020	Enhancing Two Phase-Commit Protocol for Replicated State Machines	Conference
P10	2021	Fed-agent-a Transparent ACID-Enabled Transactional Layer for Multidatabase Microservice Architectures	Conference
P11	2021	Reaching consensus in decentralized coordination of distributed microservices	Journal
P12	2022	Enhancing saga pattern for distributed transactions within a microservices architecture	Journal
P13	2022	Handling Rollbacks with Separated Response Control Service for Microservice Architecture	Conference
P14	2022	A High Availability Microservices Architecture Implementation using Saga and Backup Mechanism	Conference
P15	2023	Epoxy: ACID Transactions across Diverse Data Stores	Journal
P16	2023	ScalarDB: Universal Transaction Manager for Polystores	Journal

3.3. Step 3: Specify Criteria for Study Selection

The inclusion (IC)/ exclusion (EC) criteria were driven by two important qualities

relevance and specificity of the primary studies, these IC/EC criteria are indicated in Table 3 and were explained under subsection 4.2.1, when coming up with the initial start set.

3.4. Step 4: Identify Quality Assessment Criteria

Besides having the inclusion/exclusion criteria, it is important to establish a quality assessment (QA)criteria for all the primary studies because, among other reasons, it helps to investigate further whether quality differences explain differences in study results [13]. Our study utilized an eight-question framework for a methodical quality assessment which was based on initial quality instruments by [13] [39]. Initially, the study's purpose must be clear (Q1), with well-defined research goals. Next, a detailed depiction of the study's breadth, background, and methodological approach is necessary for a thorough comprehension (Q2). The study's impact on both scholarly inquiry and practical application must then be considered to gauge its importance (Q3). In terms of relevance, it's necessary to appraise the dependability and accuracy of the variables (Q4). Regarding rigor, it's vital for a study to respond to all the research questions (Q5), document the research methodology in a clear fashion (Q6), and distinctly convey the principal results. As a result, the above considerations would affirm their accuracy and dependability (Q7). Lastly, the study's trustworthiness is bolstered by a forthright acknowledgment of its limitations (Q8), which promotes openness and recognizes any potential limitations or biases. Employing this systematic method ensures a thorough evaluation of the literature's quality, encompassing the elements of reporting, relevance, meticulousness, and trustworthiness. The quality assessment questions used for the study are indicated in Table 4.

Table 4. Quality assessment checklist.

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

3.5. Step 5: Evaluate Quality Assessment Scores

For the assessment scale, we adopted a three-point scale (*i.e.*, yes = 1, somewhat = 0.5, no = 0). The assessment scores for each of the primary studies are provided in Table 5.

Table 5. Study quality assessment scores.

Primary Study	Reporting		Relevance		Rigor		Credibility		Total
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	
P1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
P2	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	7.5
P3	1.0	1.0	1.0	0.0	0.5	1.0	0.5	0.0	5.0
P4	1.0	1.0	1.0	0.0	0.5	1.0	0.5	0.5	5.5
P5	1.0	1.0	1.0	0.0	0.5	0.5	0.0	0.5	4.5
P6	0.5	0.0	0.5	0.5	0.5	0.0	0.5	0.5	3.0
P7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
P8	1.0	1.0	1.0	0.0	0.5	1.0	1.0	0.0	5.5
P9	1.0	1.0	0.5	0.5	0.0	1.0	0.5	0.0	4.5
P10	0.5	1.0	1.0	1.0	0.5	0.5	1.0	0.0	5.5
P11	1.0	1.0	1.0	1.0	1.0	0.5	1.0	1.0	7.5
P12	1.0	1.0	0.5	0.5	1.0	0.5	0.5	0.0	5.0
P13	1.0	0.5	0.5	0.0	0.5	0.0	0.0	0.0	2.5
P14	0.5	0.5	0.5	0.5	0.0	0.5	0.5	0.0	3.5
P15	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	8.0
P16	1.0	1.0	1.0	1.0	0.5	1.0	1.0	1.0	7.5

Findings

Reporting—Q1, and Q2: By the initial query, most studies explicitly outlined their aims in the abstract and expanded upon them in the introduction. Exceptions to this were studies P6 and P10, which did not clearly articulate their aims. In evaluating the clarity of how the studies were conducted for the second question, most studies were once again rated favorably. However, ratings were lower in instances where the context lacked a clear definition. Specifically, study P6 presented a challenge in comprehension due to grammatical errors that made it difficult to follow the logical progression of the text.

Relevance—Q3 and Q4: While all primary studies in a Systematic Literature Review (SLR) are expected to be relevant, Question 3 is designed to gauge the extent of each study's relevance. The majority of studies achieved a score of one in this regard. In terms of relevance, Question 4 aims to determine whether an evaluation was conducted and to assess the validity of the evaluation metrics. Notably, 75% of the studies [P1, P2, P6, P7, P8, P9, P10, P11, P12, P14, P15, P16] undertook some form of evaluation. Among these, the Yahoo! Cloud Serving Benchmark (YCSB) [40] was the most frequently utilized, specifically in studies P1, P10, and P16. Study P9, in particular, employed the Two-Phase Commit (2PC) protocol as a benchmark to evaluate its enhanced version of the 2PC protocol

Rigor—Q5, and Q6: Indeed, while the primary studies did not explicitly list questions, their aims inherently suggested the research questions. This inference

was then applied to evaluate the fifth quality assessment question in the context of the entire study.

Credibility—Q7 and Q8: In the context of credibility, Question 8 is particularly noteworthy as it evaluates whether each study includes an explicit acknowledgment of its limitations. It is striking to observe that approximately 50% of the studies did not provide a clear statement regarding their limitations. This omission can impact the interpretation of the studies' findings, as understanding the limitations is crucial for assessing the generalizability and applicability of the research.

3.6. Step 6: Develop Method for Data Extraction

The data extraction stage entails designing data extraction forms to be used for gathering essential information relevant to the systematic literature review (SLR) from the primary studies. Data extraction can be a time-consuming and error-prone process if not properly done [41]. One of the key steps to overcoming the highlighted challenges in [41] is to properly design the data extraction forms. In this study the extraction form was developed as guided by [13]. The form includes the study ID, title, authors, year, publication type, and summary of the primary study. To capture data relevant to this SLR, the form was expanded to include parameters on challenges of distributed transactions in MSA and a summary proposed solution. Data extraction from the primary studies was done with the aid of Mendeley Software [42]. All 16 primary studies were loaded into the software for reading and creating notes for each study by using the annotations feature. The extracted data is shown in Appendix A.

3.7. Step 7: Develop Method for Data Synthesis

Data synthesis entails the consolidation and summarizing of outcomes from primary studies within a review. For this particular review, the primary studies are predominantly descriptive, which in turn renders the synthesis process descriptive. The synthesized findings, encompassing both challenges and solutions, are detailed in Section 5 of the results.

3.8. Step 8: Define Strategy for Presenting Findings

This SLR will be submitted to a scientific journal for publication in the scientific community.

4. Results

4.1. RQ1: What Are the Challenges Associated with Managing Distributed Transactions within MSA?

The studies gathered recognize that there are indeed challenges associated with managing distributed transactions within Microservice Architectures (MSAs). These challenges have been organized into the subsequent subsections.

4.1.1. Absence of Mechanisms to Enforce Transactional Consistency across MSA with Heterogeneous Data Stores

The most common challenge identified in our primary studies was the lack of mechanisms to ensure data consistency for transactions across microservices. In his book Designing Data-Intensive Applications [4], Martin Kleppmann highlights that the term "consistency" in database systems is multifaceted, encompassing four distinct interpretations. 1) Replica consistency ensures that independent data stores reflect the most recent updates post-transaction. 2) In the CAP theorem, consistency refers to linearizability, meaning that all nodes see the same data at the same time. 3) Consistent hashing is a partitioning strategy that facilitates rebalancing in distributed systems. 4) ACID consistency pertains to a database's ability to maintain specific assertions (invariants) about the data, both before and after a transaction, as exemplified by referential integrity in traditional ACID transactions. This study primarily focuses on the last interpretation of consistency, which is also the emphasis of the primary studies reviewed. Lack of mechanisms to ensure transaction consistency in MSA was the most common challenge identified in our primary studies accounting for 56.25% of the primary studies considered. Specifically, Studies P1, P3, P4, P5, P7, P10, P12, P15 and P16 all addressed this challenge and proposed various innovative solutions. Study P3 highlights that transactions serve as a crucial abstraction for programming database applications, effectively resolving issues of consistency and concurrency that are challenging for developers to handle. Within a homogeneous distributed database system, the conventional method for executing distributed transactions is through the two-phase commit (2PC) protocol, as noted in studies P7 and P10. The 2PC protocol necessitates that databases comply with the X/Open XA [43] standard to function in a distributed system. However, this compliance requirement restricts the use of modern NoSQL databases like MongoDB, CockroachDB, and Redis, which lack XA support, thus constraining microservices reliant on these databases, as discussed in studies P15 and P16. Another drawback of the 2PC protocol is its scalability issue due to the maintenance of locks throughout its operation, which is further elaborated in studies P7 and P10. While the two-phase commit (2PC) protocol is commonly used to synchronize data and ensure high consistency, its tendency to lower throughput restricts its applicability in highly scalable microservice architectures (MSAs).

The integrity of a transaction is also at risk if it does not maintain Isolation, which aims to guarantee that each operation within a transaction occurs independently, free from the influence of other simultaneous transactions. This is crucial for averting data irregularities and upholding consistency. Study P12 highlights this issue of insufficient isolation in sagas.

4.1.2. Poor Scalability and Performance

Building on the concepts introduced in subsection 5.1.1, there is an imperative demand for enhanced scalability and performance in the realm of distributed transactions within microservices. As detailed in articles P2, P5, and P9, conventional protocols such as the Two-Phase Commit (2PC) inherently rely on locks

that are maintained throughout the transaction process. This approach markedly escalates transaction conflicts and contributes to increased latency, thus imposing scalability constraints. Consequently, there is a pressing need for more efficient transaction protocols that cater to the highly distributed nature of Microservice Architectures (MSAs), which require scalable solutions to manage their distributed transactions effectively. This particular challenge accounted for 12.5% of the identified challenges.

4.1.3. Challenges in SAGA Based Solutions

The saga design pattern is widely recognized as a pivotal strategy for managing distributed transactions within Microservice Architectures (MSAs). Defined as a transaction segmented into a series of subtransactions that can be interleaved with other transactions, sagas are instrumental in ensuring transactional integrity. They are categorized into two types:

1) Orchestration-based Sagas: These involve a central coordinator that oversees the transaction flow.

2) Choreography-based Sagas: These operate without a central coordinator, relying instead on events that are generated and handled by each participating service.

Despite their prevalence, sagas present unique challenges, which have been the subject of dynamic discussion in 25% of the studies reviewed.

Single Point of failure: Study P6 identifies the vulnerability of orchestration-based sagas to a single point of coordinator failure, which potentially compromises system availability.

Lack of rollback and compensation mechanisms: Studies P8, P13, and P14 delve into the nuances of transactional failures. Specifically, Study P13 underscores the absence of robust rollback and compensation mechanisms for rectifying failed transactions, while Study P14 zeroes in on the implications of transaction failures within orchestration-based sagas.

Coordination without Central Authority: This challenge is more prominent in choreography-based saga design patterns. A notable weakness identified in the choreography-based saga design pattern, as highlighted in Study P11, is the potential presence of cyclic dependencies between microservices. These dependencies can significantly impact the ability to achieve consensus.

4.2. RQ2: What Solutions Address the Identified Challenges?

The preceding segment recognized various obstacles encountered in handling distributed transactions within Microservice Architectures (MSA), subsequently categorizing them. This segment will explore the suggested resolutions for each challenge pinpointed. Consequently, it will be structured into corresponding categories, mirroring the challenges outlined in section 5.1.

4.2.1. Addressing Transactional Consistency

To address the lack of mechanisms for transactional consistency, several innovative

proposals and solutions have been made from the gathered studies. The solutions can roughly be categorized into two groups 1) Introduction of transactional layer and 2) modification of 2PC.

1) **Introduction of a transactional layer**

Study P3 introduces an ultra-scalable transactional management (CoherentPaas) layer that can be integrated with any data store that supports multi-versioning.

Study P4—Introduces a SagaMas framework which is a hybrid of the saga design pattern and multi-agent distributed system. The paper proposes a mechanism where each microservice will have its transactions coordinated by the agents layer leveraging Multi-Agent System to facilitate complex coordinated tasks in distributed systems, providing the necessary level of abstraction and transaction management between components. In the article "Saga Distributed Transactions Pattern" [44], several common weaknesses of the Saga design pattern are highlighted, primarily due to the lack of isolation in sagas. The article identifies challenges such as dirty reads, lost updates, and non-repeatable reads. It also suggests measures that developers can use to mitigate these issues. It is advisable for software developers researchers and solution architects to consider these challenges and the proposed workarounds from this article.

P10—Introduces Fed-Agent a transaction layer proxy between clients and microservices that coordinates read and write operations. It uses MVCC to ensure ACID guarantees and anomaly prevention in a RAFT group of nodes. It eliminates the need for additional coding by using configuration files.

P15—Proposes Epoxy, a protocol that automates transaction coordination and ensures atomicity and durability, thereby reducing the need for developers to manually handle concurrency control. Epoxy uses a primary transactional DBMS as a transaction coordinator, which initiates and manages transactions across multiple secondary data stores. Epoxy adapts Multi-Version Concurrency Control (MVCC) to work across various data stores, which helps in maintaining isolation by storing version information in record metadata and filtering reads so that transactions only see the correct versions of records. A major drawback is that once implemented, it will be the only exclusive access to the secondary store table.

Study P1 introduces the Cherry Garcia protocol which is an efficient approach for multi-item transactions across diverse key-value stores. Unlike traditional methods, it eliminates the need for a central coordinating system, simplifying data consistency management. Implemented via the Java library, Cherry Garcia provides an easy-to-use API for handling transactions across various stores. However, one limitation of Cherry Garcia is its reliance on external clocks to provide TrueTime or timestamps for snapshot isolation. This dependence can limit its applicability, as accurate timestamps may not always be available, and it introduces single points of failure within the system. Additionally, in study P1, the authors used a local clock to keep time, which would complicate the happened-before relationships when implemented globally. Another weakness of the protocol is its restriction to key-value data stores.

Study P16 ScalarDB, builds upon the Cherry Garcia protocol by utilizing a universal transaction manager that operates independently of the transaction capabilities of underlying databases.

In ScalarDB, each record in a database is treated as a separate database, performing two-phase commits over multiple records to ensure consistent transaction commits across multiple databases. ScalarDB overcomes the Cherry Garcia limitation of relying on external clocks for snapshot isolation by modifying the protocol to depend solely on linearizable operations to validate conflicting transactions. The protocol manages write-ahead logging information, referred to as disaggregated write-ahead logs, by adding transaction metadata to a record in addition to the columns managed by the application.

The authors of study P16 [45] have been commendably transparent about the protocol's weaknesses and have indicated that some of these shortcomings will be addressed in future works. The protocol requires each underlying database to provide linearizable reads and conditional mutations on a single database record, which restricts its use to database systems that meet this requirement. Additionally, transactions in ScalarDB do not support constraints such as referential integrity. Despite these weaknesses, ScalarDB is a fully-fledged production system.

2) **Modification of 2PC**

Studies P5 and P7 have approached the challenge by utilizing the optimistic concurrency control (OCC) approach. Study P5(GRIT)—leverages deterministic database technologies along with OCC protocol. In GRIT, transactions are executed optimistically, and conflict resolution occurs only at commit time. If no conflicts are detected at the commit time, the transaction is logically committed and recorded in the transaction log. Additionally, GRIT maintains a cache of recently committed transactions' write-sets to efficiently check for conflicts during execution. An article by Ren *et al.* [46] highlight a significant weakness of deterministic database technologies: the inability to arbitrarily abort transactions. This limitation leads to increased latency due to the need for a preprocessing layer that ensures identical input is sent to every replica. The authors of study P5 also acknowledge this issue.

Study P7 proposes 2PC* protocol which is an enhanced middleware approach that refines the conventional 2PC protocol. It replaces the traditional method's synchronization-blocking lock with a more efficient Secondary Asynchronous Optimistic Locking (SAOL) and further makes use of Multiversion Concurrency Control which results in a greater number of concurrent transactions. The SAOL mechanism employs dual locks, significantly boosting transaction processing velocity and efficiency while minimizing the overhead typically associated with maintaining locks during the entire transaction duration.

4.2.2. Enhancing Scalability and Performance

The studies [P2, P9] have identified that the core issue affecting the scalability and performance of distributed transactions lies in the inefficient lock management within the traditional Two-Phase Commit (2PC) protocol. Consequently, their

proposed solutions revolve around modifying the 2PC to enhance its lock management mechanisms. These modifications aim to streamline the transaction process, reduce bottlenecks, and ultimately lead to improved system performance and scalability

Study P2 improves the efficiency of the 2PC protocol by liberating write locks in the initial phase, which boosts throughput and diminishes latency. This innovative approach permits an earlier release of locks during transactions and enhances dependency tracking for superior failure management, achieving a throughput that is up to 20-fold greater than that of traditional methods. Article P9 recommends refining the 2PC protocol by adopting a priority-based system for resolving write conflicts, where higher priority values are favored. This method introduces an early termination mechanism for state executions that is triggered when a higher-priority state machine attempts to commit. This effectively reduces superfluous state executions. By ensuring that only one machine commits changes at a time, the approach decreases the frequency of failed state machine cycles and unnecessary event executions.

4.2.3. Strategies for Fault Tolerance and Effective Failure Management

Section 5.1.3 indicated that the challenges to do with fault tolerance and failure management were mostly to do with the Saga design pattern. Study P6 discussed the issue of a single point of failure for the coordinator in Orchestration sagas.

1) **Coordinator Failure**

P6 presents a fault-tolerant central saga orchestrator that enhances the reliability of microservices by managing compensations effectively in case of failures, ensuring system consistency and reliability. The proposed optimizations, such as storing only essential data and failed saga data, significantly reduce memory and time requirements. The orchestrator is built as a cluster that sends HTTP requests to idempotent endpoints, which helps in maintaining system availability and reducing the impact of failure.

2) **Transaction Failure**

P8 Buffered serialization improves transaction consistency by caching transaction metadata in a platform-agnostic format, allowing the system to retry and restore transactions in case of failures, ensuring data consistency across services. The buffered serialization approach ensures that transaction data is available for recovery, which means that even if a local transaction fails, the system can use the cached metadata to restore and maintain a consistent state. Delegating the serialization and deserialization tasks to a middle-tier buffer reduces the overhead on individual microservices.

P13 Suggests employing a separate control microservice that monitors the microservices response queue and determines whether a rollback is warranted. This isolation enables microservices to concentrate on their specific tasks, resulting in efficient CPU and memory utilization. Furthermore, parallel execution of microservices would enhance system safety and speed. P14 proposes a library called Anser-Saga to help developers manage data consistency in microservices by creating

backups and restart points for distributed transactions, ensuring that data remains consistent even if a failure occurs during the process. The backup mechanism in Anser-Saga allows for the rapid restart of transactions if the central orchestrator fails.

4.2.4. Decentralized Coordination Strategies

P11 Proposes decentralized microservice compositions where participants can reach consensus at runtime, reducing the dependency on a central coordinator and thus mitigating isolation issues. Further highlights mechanisms where services execute compensations and cancel resource reservations after timeout periods if no confirmation or cancellation is received.

5. Discussion

The systematic literature review (SLR) reveals a comprehensive landscape of challenges and solutions in the realm of distributed transactions within microservices architectures. For developers and software architects to effectively design and implement microservices solutions, it is crucial to understand these challenges and adopt the identified solutions or develop new ones.

5.1. Identified Challenges

The primary studies highlighted various challenges encountered in the management of distributed transactions within MSA. The majority of the studies brought out the absence of mechanisms to enforce consistency across microservices that are backed by some heterogeneous databases. The traditional 2PC is inadequate in heterogeneous database systems due to a lack of X/Open XA support from the NoSQL family of databases. Further, 2PC uses locking mechanisms to enforce data isolation which, however, affects the scalability of distributed transactions and lowers their general throughput. Other studies brought out the challenges faced in SAGA-based distributed transactions management solutions. A single point of failure is one of the challenges in an Orchestration-based solution. Others were a lack of rollback and compensation mechanisms for rectifying failed transactions. Choregraphy-based SAGA solutions have the potential challenge of creating cyclic dependencies among the microservices.

5.2. Proposed Solutions

To address these challenges, the studies proposed various solutions, including:

1) Transactional Management Layers: In responding to the challenge of ensuring data integrity and consistency in the distributed transactions of MSA, the majority of the studies propose/implement an independent transaction management layer. P15 introduces Epoxy a protocol that adapts Multi-Version Concurrency Control (MVCC) to work across various data stores, which helps in maintaining isolation by storing version information in record metadata and filtering reads so that transactions only see the correct versions, whereas Study P3 introduces a transactional layer called CoherentPass. Studies P1 and P16 utilize the Cherry

Garcia transaction management protocol which uses Snapshot Isolation thereby simplifying the handling of distributed transactions. Cherry Garcia employs a client-coordinated transaction commitment protocol that boosts scalability and fault tolerance while maintaining data consistency across various data stores. This Java library is designed to facilitate multi-item transactions across heterogeneous data stores, enabling efficient transaction management without the need for a central coordinating infrastructure. It's noteworthy to emphasize that ScaladDB is a fully functional software at the production stage with several clients utilizing the product [47]. Having a product that is at the production stage with an active community of clients utilizing it, speaks to the pragmatism of the approach to introduce a management layer on top of the traditional database management systems. Study P4 equally proposes a transaction management layer called SagaMas which utilizes multi-agent systems.

2) Enhanced Protocols: Modifying existing protocols, such as releasing write locks during phase one of the Two-Phase Commit (2PC) protocol to improve throughput and reduce latency. Studies P5 and P7 are typical examples of those advocating for the enhancement of 2PC.

3) Frameworks for Sagas: Developing frameworks like SagaMAS-study P4 to better manage distributed transactions in a microservices architecture, addressing the hidden nature of low-level events in Saga choreography.

5.3. Implications for Future Research

The findings of this SLR underscore the need for ongoing research and development in the field of managing distributed transactions in MSA. The evolution of microservices and their widespread adoption in various industries necessitate robust and scalable solutions for managing distributed transactions. P1 and P7 intend to extend their implementations to other data stores, whereas P8 plans to extend its application to areas like cloud, mobile, and IoT environments. P3 and P13 evaluate the performance of their proposed solutions as part of their future works. P16 suggested several future research areas including enabling the transaction manager to handle more database abstractions without complicating it. However, since scalardb is proprietary the proposed future research areas can only be done in-house. In general, open areas for future research include the following:

1) Developing comprehensive frameworks: There is a need for more comprehensive frameworks that can seamlessly integrate with various microservices architectures and provide robust transaction management. Study P4 [48] by Xavier Limon *et al.* proposes the introduction of a multi-agent-based framework called SagaMAS which acts as an autonomous layer that coordinates distributed transactions, simplifying interactions between microservices and relieving developers from transaction management tasks. Multi-agent systems (MAS) have a wide number of successful application areas, including but not limited to modeling complex systems, smart grids, computer networks, civil engineering and distributed artificial intelligence [49]. The article [48] therefore envisions the possibility of combining the versatile MAS with microservices to resolve the challenge of

distributed transactions in MSA. The general idea in SagaMas is that each microservice has an associated agent, which handles its transactions in coordination with other agents. In developing the Sagamas framework the authors adopted the use of the Prometheus software methodology for software development, which is further divided into three (3) major stages a) system specification b) architectural design, and c) detailed design. They worked on stage d) the system design stage, leaving the other two open for future research works.

2) Improving efficiency and performance: P2 proposes performance improvement of its solutions by trying out techniques such as multi-version optimistic concurrency control so that transactions can read or overwrite data that was last written by a still active transaction. P10 also highlights improving performance for example by splitting the system into shared-nothing partitions among other techniques.

3) Benchmarking and evaluation: Each of the primary studies considered had its own benchmarking and evaluation methodology and approach. This poses a great challenge when it comes to comparing the "apples-to-apples" performance of each considered solution. Moreover, synthesizing the data from the primary studies equally becomes problematic. We can take a leaf from traditional data management platforms that are evaluated using industry-standard benchmarks such as TPC-C and TPC-E [50]. We highlight some of the notable evaluation methods used, studies P15 used an adapted version of TPC-C, and P10 evaluated using YCSB [40]. YCSB is more specifically designed for evaluating NoSQL databases. P16 utilized both TPC-C and YCSB. P1 utilized YCSB+ [51], an extension of YCSB, that wraps database operations within transactions and is better suited for distributed transaction operations. It is clear from the foregoing that there is a great need, therefore, to come up with some common industrial benchmark and methodology for assessing heterogeneous distributed database systems in microservices architecture. Some future research can focus on the development of standardized benchmarks and evaluation protocols for ensuring the comparability of results across different studies.

4) Addressing limitations: Almost 50% of the considered studies did not explicitly state their limitations as indicated in **Table 5** focussing on question 8. This tends to affect the credibility of the studies. We encourage researchers in their future studies to consider explicitly acknowledging their limitations, which would enhance the credibility and applicability of their findings.

By addressing these areas, researchers can contribute to the development of more secure and efficient microservices architectures, ultimately improving the reliability and scalability of distributed systems.

5.4. Threats to Validity

Kitchenham's guidelines [13] emphasize the importance of safeguarding against bias and ensuring validity throughout key stages of the Systematic Literature Review (SLR) process. For SLR conducted as part of graduate research studies, it is

recommended that the primary researcher actively engages with their supervisor to receive feedback and critique. The essential steps requiring such oversight include:

1) Protocol Review: Ensuring the research protocol is thoroughly vetted.

2) Evaluating Quality Assessment: Evaluating how the scores were assigned to each question per primary study

3) Data Extraction: Carefully extracting data in a consistent and unbiased manner

Brereton *et al.* [52] also advocate for a collaborative approach during the data extraction phase of the SLR, suggesting a division of roles where one author extracts data while another reviews it. Consistent with these recommendations, the supervisor, serving as the co-author, played a pivotal role in the planning, execution, and documentation of the SLR. This collaborative approach helps maintain the integrity and quality of the research.

6. Conclusions

This systematic literature review provides a comprehensive analysis of the state-of-the-art in managing distributed transactions within microservices architectures (MSA). The review covers a decade of research from 2014 to 2024, focusing on the key challenges, proposed solutions, and emerging trends in this dynamically evolving field.

The study brought out some challenges in the management of distributed transactions. Foremost, is the complexity of ensuring data consistency and integrity across multiple microservices and heterogeneous data stores. Second, Limitations of traditional transaction protocols like Two-Phase Commit (2PC) in terms of scalability and latency. Lastly, the need for robust mechanisms to handle concurrency and fault recovery in distributed environments.

The proposed solutions include the Implementation of client-side libraries, such as Cherry Garcia, to manage transactions using Snapshot Isolation. Modifications to existing protocols, including early release of write locks during the first phase of 2PC, to improve throughput and reduce latency. Introduction of layers like CoherentPaas to ensure data integrity and consistency in polyglot persistence environments. Development of frameworks like SagaMAS to manage distributed transactions, providing the necessary abstraction for communication and coordination between microservices.

The findings underscore the need for ongoing research and development to address the evolving challenges in managing distributed transactions within microservices architectures. Creating robust frameworks that integrate smoothly with diverse MSA and provide reliable transaction management. Explicitly acknowledging and addressing the limitations of current solutions to enhance their applicability and credibility. Standardizing benchmarks and evaluation protocols to ensure the comparability of results across different studies.

This review highlights the critical importance of effective transaction ma-

nagement in the successful deployment and operation of microservices architectures. By shedding light on the current landscape and identifying areas for further investigation, this study aims to equip practitioners and researchers with valuable insights into the latest advancements and best practices in this field. The ongoing discourse and research efforts will undoubtedly contribute to the development of more secure, efficient, and scalable microservices-based systems, ultimately enhancing the reliability and performance of distributed computing environments.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Richardson, C. (2018) Microservices Patterns: With Examples in Java. Simon and Schuster.

[2] Fielding, R.T. (2000) Rest: Architectural Styles and the Design of Network-Based Software Architectures. Doctoral Dissertation, University of California.

[3] Taibi, D., Lenarduzzi, V., Pahl, C. and Janes, A. (2017) Microservices in Agile Software Development. *Proceedings of the XP*2017 *Scientific Workshops*, Cologne, 22-26 May 2017, 1-5. https://doi.org/10.1145/3120459.3120483

[4] Kleppmann, M. (2017) Designing Data-Intensive Applications: The Big Ideas behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, Inc.

[5] Haerder, T. and Reuter, A. (1983) Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys*, **15**, 287-317. https://doi.org/10.1145/289.291

[6] Carrera-Rivera, A., Ochoa, W., Larrinaga, F. and Lasa, G. (2022) How-to Conduct a Systematic Literature Review: A Quick Guide for Computer Science Research. *MethodsX*, **9**, Article 101895. https://doi.org/10.1016/j.mex.2022.101895

[7] Lewis, J. and Fowler, M. (2014) Microservices, a Definition of This New Architectural Term.

[8] Pahl, C. and Jamshidi, P. (2016) Microservices: A Systematic Mapping Study. *Proceedings of the* 6*th International Conference on Cloud Computing and Services Science*, Rome, 23-25 April 2016, 137-146. https://doi.org/10.5220/0005785501370146

[9] Vural, H., Koyuncu, M. and Guney, S. (2017) A Systematic Literature Review on Microservices. In: Goos, G. and Hartmanis, J., Eds., *Lecture Notes in Computer Science*, Springer, 203-217. https://doi.org/10.1007/978-3-319-62407-5_14

[10] Karabey Aksakalli, I., Çelik, T., Can, A.B. and Teki̇nerdoğan, B. (2021) Deployment and Communication Patterns in Microservice Architectures: A Systematic Literature Review. *Journal of Systems and Software*, **180**, Article 111014. https://doi.org/10.1016/j.jss.2021.111014

[11] Laigner, R., Zhou, Y., Salles, M.A.V., Liu, Y. and Kalinowski, M. (2021) Data Management in Microservices. *Proceedings of the VLDB Endowment*, **14**, 3348-3361. https://doi.org/10.14778/3484224.3484232

[12] Berardi, D., Giallorenzo, S., Mauro, J., Melis, A., Montesi, F. and Prandini, M. (2022) Microservice Security: A Systematic Literature Review. *Peer J Computer Science*, **7**, e779. https://doi.org/10.7717/peerj-cs.779

[13] Keele, S., *et al.* (2007) Guidelines for Performing Systematic Literature Reviews in Software Engineering.

[14] Sedeño, J., Vázquez, G., Escalona, M.J. and Mejías, M. (2019) The Systematic Discovery of Services in Early Stages of Agile Developments: A Systematic Literature Review. *Journal of Computer and Communications*, **7**, 114-134. https://doi.org/10.4236/jcc.2019.77012

[15] Saeed, W. and Omlin, C. (2023) Explainable AI (XAI): A Systematic Meta-Survey of Current Challenges and Future Opportunities. *Knowledge-Based Systems*, **263**, Article 110273. https://doi.org/10.1016/j.knosys.2023.110273

[16] Mittal, M., Kumar, K. and Behal, S. (2022) Deep Learning Approaches for Detecting Ddos Attacks: A Systematic Review. *Soft Computing*, **27**, 13039-13075. https://doi.org/10.1007/s00500-021-06608-1

[17] Tiwari, S., Al-Aswadi, F.N. and Gaurav, D. (2021) Recent Trends in Knowledge Graphs: Theory and Practice. *Soft Computing*, **25**, 8337-8355. https://doi.org/10.1007/s00500-021-05756-8

[18] Quin, F., Weyns, D., Galster, M. and Silva, C.C. (2024) A/B Testing: A Systematic Literature Review. *Journal of Systems and Software*, **211**, Article 112011. https://doi.org/10.1016/j.jss.2024.112011

[19] Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J. and Linkman, S. (2009) Systematic Literature Reviews in Software Engineering—A Systematic Literature Review. *Information and Software Technology*, **51**, 7-15. https://doi.org/10.1016/j.infsof.2008.09.009

[20] Wohlin, C. (2014) Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. *Proceedings of the* 18*th International Conference on Evaluation and Assessment in Software Engineering*, London, 13-14 May 2014, 1-10. https://doi.org/10.1145/2601248.2601268

[21] Wohlin, C., Kalinowski, M., Romero Felizardo, K. and Mendes, E. (2022) Successful Combination of Database Search and Snowballing for Identification of Primary Studies in Systematic Literature Studies. *Information and Software Technology*, **147**, Article 106908. https://doi.org/10.1016/j.infsof.2022.106908

[22] Halevi, G., Moed, H. and Bar-Ilan, J. (2017) Suitability of Google Scholar as a Source of Scientific Information and as a Source of Data for Scientific Evaluation—Review of the Literature. *Journal of Informetrics*, **11**, 823-834. https://doi.org/10.1016/j.joi.2017.06.005

[23] Cusker, J. (2013) Elsevier Compendex and Google Scholar: A Quantitative Comparison of Two Resources for Engineering Research and an Update to Prior Comparisons. *The Journal of Academic Librarianship*, **39**, 241-243. https://doi.org/10.1016/j.acalib.2013.02.001

[24] Fan, W., Han, Z., Zhang, Y. and Wang, R. (2018) Method of Maintaining Data Consistency in Microservice Architecture. 2018 *IEEE* 4*th International Conference on Big Data Security on Cloud* (*Big Data Security*), *IEEE International Conference on High Performance and Smart Computing*, (*HPSC*) *and IEEE International Conference on Intelligent Data and Security* (*IDS*), Omaha, 3-5 May 2018, 47-50. https://doi.org/10.1109/bds/hpsc/ids18.2018.00023

[25] Rudrabhatla, C.K. (2018) Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture. *International Journal of Advanced Computer Science and Applications*, **9**, 18-22. https://doi.org/10.14569/ijacsa.2018.090804

[26] Štefanko, M., Chaloupka, O. and Rossi, B. (2019) The Saga Pattern in a Reactive Microservices Environment. *Proceedings of the* 14*th International Conference on Software Technologies*, Setubal, 26-28 July 2019, 483-490. https://doi.org/10.5220/0007918704830490

[27] Malyuga, K., Perl, O., Slapoguzov, A. and Perl, I. (2020) Fault Tolerant Central Saga

Orchestrator in Restful Architecture. 2020 26*th Conference of Open Innovations Association*, Yaroslavl, 20-24 April 2020, 278-283. https://doi.org/10.23919/fruct48808.2020.9087389

[28] Žežulka, M., Chaloupka, O. and Rossi, B. (2021) Integrating Distributed Tracing into the Narayana Transaction Manager. *Proceedings of the* 6*th International Conference on Complexity, Future Information Systems and Risk*, 24-25 April 2021, 55-62. https://doi.org/10.5220/0010448200550062

[29] Ogura, T., Akita, Y., Miyazawa, Y. and Kawashima, H. (2021) Accelerating Geo-Distributed Transaction Processing with Fast Logging. 2021 *IEEE International Conference on Big Data*, Orlando, 15-18 December 2021, 2390-2399. https://doi.org/10.1109/bigdata52589.2021.9671560

[30] Laigner, R., Zhou, Y. and Salles, M.A.V. (2021) A Distributed Database System for Event-Based Microservices. *Proceedings of the* 15*th ACM International Conference on Distributed and Event-Based Systems*, Italy, 28 June-2 July 2021, 25-30. https://doi.org/10.1145/3465480.3466919

[31] Aydin, S. and Cebi, C.B. (2022) Comparison of Choreography vs Orchestration Based Saga Patterns in Microservices. 2022 *International Conference on Electrical, Computer and Energy Technologies*, Prague, 20-22 July 2022, 1-6. https://doi.org/10.1109/icecet55527.2022.9872665

[32] Djerou, M. and Tibermacine, O. (2022) SAGA Distributed Transactions Verification Using Maude. 2022 International Symposium on iNnovative Informatics of Biskra, Biskra, 7-8 December 2022, 1-6. https://doi.org/10.1109/isnib57382.2022.10076050

[33] de Heus, M., Psarakis, K., Fragkoulis, M. and Katsifodimos, A. (2022) Transactions across Serverless Functions Leveraging Stateful Dataflows. *Information Systems*, **108**, Article 102015. https://doi.org/10.1016/j.is.2022.102015

[34] Ishida, A., Katsuno, Y., Tozawa, A. and Saito, S. (2023) Automatically Refactoring Application Transactions for Microservice-Oriented Architecture. 2023 *IEEE International Conference on Software Services Engineering*, Chicago, 2-8 July 2023, 210-219. https://doi.org/10.1109/sse60056.2023.00035

[35] González-Aparicio, M.T., Younas, M., Tuya, J. and Casado, R. (2023) A Transaction Platform for Microservices-Based Big Data Systems. *Simulation Modelling Practice and Theory*, **123**, Article 102709. https://doi.org/10.1016/j.simpat.2022.102709

[36] Koutanov, E. (2023) Strict Serializable Multi-Database Certification with Out-of-Order Updates. Authorea Preprints.

[37] Reza, K. and Rahman, N. (2022) Explication and Extension of Saga and Microservice Patterns to Enable Resilient Distributed Transaction. In: Ranganathan, G., Fernando, X. and Rocha, A., Eds., *Inventive Communication and Computational Technologies*, Springer, 209-220. https://doi.org/10.1007/978-981-19-4960-9_18

[38] Fan, P., Liu, J., Yin, W., Wang, H., Chen, X. and Sun, H. (2020) 2PC+: A High Performance Protocol for Distributed Transactions of Micro-Service Architecture. In: Gao, H.H. and Yin, Y.Y., Eds., *Intelligent Mobile Service Computing*, Springer, 93-105. https://doi.org/10.1007/978-3-030-50184-6_6

[39] Kitchenham, B. and Brereton, P. (2013) A Systematic Review of Systematic Review Process Research in Software Engineering. *Information and Software Technology*, **55**, 2049-2075. https://doi.org/10.1016/j.infsof.2013.07.010

[40] Yahoo (2024) Title of Webpage. https://research.yahoo.com/news/yahoo-cloud-serving-benchmark

[41] Garousi, V. and Felderer, M. (2017) Experience-Based Guidelines for Effective and

Efficient Data Extraction in Systematic Reviews in Software Engineering. *Proceedings of the* 21*st International Conference on Evaluation and Assessment in Software Engineering*, Karlskrona, 15-16 June 2017, 170-179. https://doi.org/10.1145/3084226.3084238

[42] Mendeley Ltd. (2024) Mendeley Reference Manager. Version 2.68.0. https://www.mendeley.com/reference-management/reference-manager/

[43] Open Group Library (2024) Distributed Transaction Processing: The XA Specification. https://pubs.opengroup.org/onlinepubs/009680699/toc.pdf

[44] Microsoft Learn (2024) Saga Distributed Transactions Pattern. https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga

[45] Yamada, H., Suzuki, T., Ito, Y. and Nemoto, J. (2023) Scalardb: Universal Transaction Manager for Polystores. *Proceedings of the VLDB Endowment*, **16**, 3768-3780. https://doi.org/10.14778/3611540.3611563

[46] Ren, K., Thomson, A. and Abadi, D.J. (2014) An Evaluation of the Advantages and Disadvantages of Deterministic Database Systems. *Proceedings of the VLDB Endowment*, **7**, 821-832. https://doi.org/10.14778/2732951.2732955

[47] Scalar (2024) ScalarDB. https://www.scalar-labs.com/scalardb

[48] Limon, X., Guerra-Hernandez, A., Sanchez-Garcia, A.J. and Perez Arriaga, J.C. (2018) SagaMAS: A Software Framework for Distributed Transactions in the Microservice Architecture. 2018 6*th International Conference in Software Engineering Research and Innovation*, San Luis Potosi, 24-26 October 2018, 50-58. https://doi.org/10.1109/conisoft.2018.8645853

[49] Dorri, A., Kanhere, S.S. and Jurdak, R. (2018) Multi-Agent Systems: A Survey. *IEEE Access*, **6**, 28573-28593. https://doi.org/10.1109/access.2018.2831228

[50] TPC. (2024) TPC Specifications. https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp

[51] Dey, A., Fekete, A., Nambiar, R. and Rohm, U. (2014) YCSB+T: Benchmarking Web-Scale Transactional Databases. 2014 *IEEE* 30*th International Conference on Data Engineering Workshops*, Chicago, 31 March-4 April 2014, 223-230. https://doi.org/10.1109/icdew.2014.6818330

[52] Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M. and Khalil, M. (2007) Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain. *Journal of Systems and Software*, **80**, 571-583. https://doi.org/10.1016/j.jss.2006.07.009.