Microservices in Practice: A Systematic Literature Review

Shubham Verma*,Abhishek Kumar Rai†,Parag Gupta‡, Gaurav Parashar¶
Department of Computer Science and Engineering
KIET Group of Institutions
Ghaziabad, India
Email: *shubham.2125cse11599@kiet.edu,
†abhishek.2125cse1072@kiet.edu, §parag.2125ec1112@kiet.edu,
¶gauravparashar24@gmail.com (ORCID: 0000-0003-4869-1819)

***Abstract*—Microservice is an architectural style where the application is structured as a combination of different small independent services, each catering to different to a specific business solution. In this paper, we did a systematic literature review of different papers related to the microservice architecture using the Kitchenham systematic literature survey. We extracted different microservice architectures, their limitations, and future scope.**

***Index Terms*—Microservices, microservice architecture, microservices limitations**

I. Introduction

Web development is a not a new concept. It has been there for a long time. But it has evolved over the time. The ways websites were built and handled in earlier has also evolved with the expansion and availability of internet. The types of users and user experience has changed. Earlier companies used to follow monolithic architecture [1] for developing web applications and they were doing good with that. But the user base of companies kept growing so, there were some serious problems that were to be addressed in order to handle the increasing users and scalability of the architecture. Scalability, slow development, couplings, size and complexity, single point of failures, cascading failures are some common problem with monolithic architecture. Monolithic applications have their limitation, and because of that they cannot be scaled beyond a limit [1]

Microservices architecture [2] in contrast to monolithic architecture [2] [1] has the ability to easily adapt according to the changing requirements by separating the concerns and dividing them across the decomposed services which helps in faster development, scalability, resiliency, and avoiding single point of failures.

Companies started moving towards the microservices architecture that promises to address the problems with the monolithic architecture. Companies like Netflix, Coca Cola, Uber and Amazon are some of the early adopters of microservice architecture. Netflix adopted this architecture even before it was properly introduced in around 2009. But it's not all that straight forward. There are several other problems that arise while implementing this architecture despite its promises. Due to it's distributed nature it inevitability brings the complexities and problems of distributed systems with it.

If any company is trying to move from monolithic architecture to microservice architecture, they have to research and rectify existing issues and challenges. In this paper, we are trying to present the possibilities and limitations of this architecture so that one can decide what are the things that are in favour and what are not.

II. Background and Related Work

Uber, Netflix, Amazon, Coca Cola, Spotify are some of the big names that support microservice architecture by accepting its limitations and has been working on them from a long time. Talking about Netflix, it is one of the early adopters of this architecture even though there was a lots of criticism around that. Netflix had to take that move due to its increasing data and user information which was becoming difficult to handle with a monolithic architecture and the serious issues with their own data centres. Finally, Netflix had to move on to cloud and for that Netflix had to split it's single monolithic application into hundreds of small loosely coupled services in order to expand and scale. This transition allowed Netflix to significantly enhance development and deployment of its platform and services. Netflix openly supports this architecture and even have lot of open source tools favouring the same.[1]

Even Uber faced similar issues with it's monolithic architecture which prevented the teams to operate independently. They initially had two monolithic applications that caused many operational issues [2] (like Availability Risks, Poor separation of concerns, Risky and expensive deployments, Inefficient execution) for them and hence they adopted microservice architecture.

Microservice architecture has evolved a lot over time and this is not the first of it's kind. Before this the industry was following another similar architecture SOA (Software-oriented Architecture) but it was not beneficial for every level of organization due to it's own limitations. Microservices focuses more on independency, decoupling, easy deployment, high

[1]https://netflixtechblog.com/tagged/microservices
[2]https://eng.uber.com/microservice-architecture/

scalability unlike SOA. Though SOA has provided significant support to the microservice architecture for its evolution, but both have different use cases.[3]

For this paper we had reviewed many different research papers from Scopus indexed journals and conferences. Organized the paper as follows: in Section III, we describe the methodology used to search and collect related papers for this study. In Section IV, we describe the microservice architecture, its properties and different architectural patterns. Section V describes the several limitations of microservice architecture which need to be addressed before implementation. In Section VI we explained and suggested our recommendations for future researchers based on our analysis.

III. Methodology

This section describes the Kitchenham systematic literature survey [3] [4] adopted to study the microservices architecture. The review is based on evidence based software engineering methodology.

A. Aims and Research Questions

In this research our main aim is to find the scenarios where microservices can be best utilized to replace the existing monolithic architecture.

Our research addresses the following three questions:

RQ1 : *What are the available architectures of microservices?*

RQ2 : *What are the limitations of existing microservices architectures?*

RQ3 : *Are there any solutions or suggestions for overcoming the challenges in the microservices architectures, from the experts?*

RQ4 : *How different companies have been impacted by adopting the microservices? Is the impact positive or negative?*

B. Search and Selection Process

The search criteria or keywords used were as follows:

- For our research we have utilized the tool Harzing's Publish or Perish. The paper's publishing year falls between 2017 and 2021 and the source was taken as Scopus.
- Using this tool we were able to find 200 papers using the keywords microservice architectures and using the keywords microservice limitations we are able to find another 57 papers.
- After removing some duplicate papers we were left with a total of 250 papers.
- Out of total 250 items there were 91 Articles, 3 Book Chapters, 151 Conference Papers and 5 Review Papers.

C. Inclusion and Exclusion Criteria

The main aim of this research is to identify and classify papers related to architectural limitations with microservices and about different types of microservice architectures. Whatever matched and aligned with our research questions we included all those papers. In this step, we have excluded all those papers that were irrelevant in the study.

Inclusion: Papers, books, white papers, periodicals from Nature, IEEE transactions, International Conferences, Journals, ACM, SVN, Arxiv papers must include microservice, paper must refer microservice architecture. The language of the paper must be English.

Exclusion: Papers other than the list of inclusion are excluded, papers not related to the microservices domain in web development specifically, papers not directly concerning our research like papers about monolithic architecture, microservices in IOT systems, and papers not in English language.

Total 45 papers were selected finally by excluding the papers based on exclusion criteria.

D. Quality Assessment

For the proper quality check we have assessed the papers on the following checkpoints:

1) Is the paper based on research (or on expert opinion)? Yes / Partly / No - 1, 0.5, 0
2) Is the aim of study clear? Yes / Partly / No - 1, 0.5, 0
3) Is the study providing any value? Yes / Partly / No - 1, 0.5, 0
4) Is there any clear findings? Yes / Partly /No - 1, 0.5, 0
5) Is the description of the context of the research adequate to carry out study? Yes / Partly / No - 1, 0.5, 0

We calculated the quality score and selected the papers as per the best scores only.

E. Data Collection

Based on the research papers, we got from our search & selection process and after applying the inclusion and exclusion criteria, we clustered the papers into various types of articles (Refer table I and table II).

The data extracted from all those 45 papers were:

- Journal name and reference
- Classification of the study
- Abstract and conclusion of the research
- Quality assessment scoring
- Research question answer

Researchers extracted the date and compiled it for further investigation.

F. Data Analysis

From all the finalized papers, some of them, that are talking about the main points about the microservice architecture are described here with the findings and comments that are extracted from the papers: Refer table III

G. Deviations from Protocol

Things that we have studied beyond our research questions in order to complete the research like about monolithic architectures. Flows including different parts of the applications can be executed by monolithic structure in easier way, since all pertinent data is in one easy-to-get place.

[3]https://www.ibm.com/cloud/blog/soa-vs-microservices

TABLE I
DIFFERENT INTENTS OF PAPERS BASED ON ARTICLE TYPES

Article Types	Count	Intent of Paper
Journal Papers	13	Review, microservices, migration patterns, virtualization, domain-driven microservices, etc
Book Chapters	1	Microservices, migration from monolithic to microservice
Conference Papers	29	Microservices applications, Bad smells, Cloud, containerization, microservices and architecture, migration to microservices, reviews, etc.
Reviews	2	Cloud, microservices, microservice applications, Web services

TABLE II
CLASSIFICATION OF PAPERS BASED ON DATA COLLECTED

Classification	Count
Cloud	15
Containerization	8
DevOps	4
Domain-driven microservices	2
Microservice and monoliths	2
Microservices	45
Microservices applications	17
Microservice architecture	28
Migration to microservices	10
Reviews	4
Security	5

H. Search Results

We found the following search results for our research questions:

- **RQ1:** Nowadays, monolithic architectures are moving rapidly towards microservices architectures, and they are providing more flexible, scalable, and efficient systems [1]. In load testing scenarios, monolithic architectures provide slightly higher performance than microservice architectures [10], but microservices architecture is being used to maintain consistency and availability of services for various business needs [6]. Applications based on microservices are easier to build and maintain since microservices are small and can be deployed independently through lightweight mechanisms [7]. Key characteristics include communication use of endpoints, componentization, and organization of business capabilities [5]. Communication protocols and containerization are involved in the implementation [17]. Apart from the advantages of adaptability and scalability, the microservices architecture needs careful measures to avoid issues in design and implementation [9][8].
- **RQ2:** Apart from the advantages of microservices architecture, it has several limitations and challenges. In open systems where microservices can join or leave arbitrarily, trust management is one of the major concerns [12]. The existing trust models are not tailored for open systems and they are more domain-specific, requiring more adaptable solutions [11]. Security is also one of the major concerns, as layered solutions are not readily available [9]. Designing microservices-based systems is not an easy task; practitioners face challenges in designing, developing, and maintaining [7]. All these provide both opportunities and challenges, but it is not clear how to follow the best practices [5]. However, if security challenges are addressed, microservices can potentially improve system robustness through loose coupling and isolation [10]. Thus, due to these limitations, microservices architecture needs more research and development [6].
- **RQ3:** The distributed nature of microservices architecture (MSA) creates special issues in a variety of fields. To guarantee scalability, high availability, and low latency, service discovery needs to be carefully planned and implemented [17]. Distributed transactions, backup plans, and data consistency are all complicated aspects of data management, which emphasizes availability and integration trade-offs [5]. Diverse technologies and the requirement for automated regression, performance, acceptance, and resilience testing in agile environments make MSA testing challenging [6]. To fulfill quality standards, performance optimization requires proactive measurement, prediction, and improvement [21]. To manage dependability and intricate workflows, communication and integration require strong, lightweight protocols [20]. Scalability, resource allocation, deployment, load balancing, persistent

TABLE III
SUMMARY: ANALYSIS OF DATA GATHERED FROM THE PROCESS

Authors	YOP	Findings	Comments	Classification of the study	QA Score
Saša Baškarada et al. [5]	2018	Microservice is a promising architecture but does not solves every problem and have its own limitations	Microservice architecture cannot be adopted by everyone. Some may find it easier while other's may struggle to adopt it.	Microservies, DevOps	4.0
Paolo Di Francesco et al. [6]	2019	This study is intended to provide both researchers and practitioners with a comprehensive picture of the field's current trends, implications and potential for industrial adoption	There still exists a big trade off between flexibility and complexity in microservices which is a main research area	Microservices, microservice architecture	3.5
Davide Taibi et al. [7]	2018	This study characterise several microservice architectural style patterns as well as the concepts that influence their definition. Different architecture patterns develop for different migration, orchestration, storage, and deployment settings, we can conclude.	Microservices must be addressed in conjunction with their deployment via containers or virtual machines, according to the patterns	DevOps, Cloud, Microservices Architecture	4
Miika Kalske et al. [8]	2017	Complexity, scalability, and code ownership are all issues that software organizations face. Even though the change has its own set of obstacles, they may be less difficult to overcome than those encountered by monolithic corporations	Microservice architecture refactoring can take many years and needs buy-in from all levels of the business. This isn't a perfect solution that will work for every company.	microservices applications, migration to microservices	4.5
Javad Ghofrani and Daniel Lübke [9]	2018	This study concerns the design and development process of microservices and the major drivers for and against using systematic approaches in microservices architectures	Optimization in Response time, Performance, and Security have higher priorities than Memory Utilization, Resilience, and Fault Tolerance.	Microservices	3.5
He Zhang et al. [10]	2019	Organizational change, decomposition, distributed monitoring, and bug-spotting are just a few of the issues that have been mentioned.	Advantages of microservices may be achieved via practise, as well as their potential drawbacks that must be handled.	Security, reviews	4
Armin Balalaie et al. [11]	2018	This study identifies different migration and rearchitecting design patterns that are collected from industrial-scale software migration projects which helps organization in developing effective and efficient migration planning	Development of pattern language which enables automated pattern composition is an area of research.	microservices architecture, migration to microservices	4
Nicola Dragoni et al. [12]	2017	Microservices architecture [2] is build on some basic principles: Bounded context, Size and Independency. Distribution, Scalability, Portability, Elasticity, Availability and Robustness are some advantages of microservice that this architecture accounts on.	Despite it's feature, the security perspective the still the main concern. Low computation devices are at higher risk from security perspective.	Microservice Architecture, security, migration to microservice	4.5

storage, and failure recovery are challenges for service orchestration [7][12]. Security is still crucial, despite the difficulties with lightweight frameworks, traffic monitoring, and access control [10]. Monitoring, tracing, and logging demand cohesive systems for anomaly detection and troubleshooting [23]. Decomposition further requires precise functional separation to achieve cohesive, loosely coupled services, failing which scalability and performance suffer [9].

- **RQ4:** Adoption of microservices has had both beneficial and detrimental effects on various businesses. After switching from monolithic to microservices designs, several organizations have seen improvements in scalability and maintainability [15][5]. Microservices have been successfully used by major online retailers like Amazon and Netflix to advance corporate goals and improve system adaptability [10][17]. Operational and cultural barriers are among the difficulties that come with the adoption process [24]. Although maintainability is frequently improved, security issues have been brought to light [9]. Although the degree of automation and DevOps processes vary from company to company, the influence on software quality is typically seen as favorable [14][12]. It's interesting to note that early adopters prioritize the advantages of scalability, whilst established businesses typically concentrate on maintainability enhancements [7]. All things considered, adopting microservices necessitates giving significant thought to the unique requirements and difficulties of a company [24].
  Using containerization comes to rescue when it comes to easy deployment of lots of microservices and their handling, monitoring them regularly and may big companies like AWS, DigitalOcean, Infosys, Intel, Tesla, Google, etc have adopted microservices along with containerization because of the benefits of containerization with microservices.
  According to the paper [13], microservices have received mostly positive or neutral response from the industry, however security is considered to be the negative factor. Also, harsh reality is, even when most of the big tech companies are moving to micorservice architecture with a major success, we must also ask a question before taking this step, do we actually need microservice based on the size of current market of the product, size of organization,

and are you ready to deal with costs and pitfalls of this architecture? Moving monolithic to microservices is a costly and time demanding operation. [14]. For most of the small organizations to start with, monolithic architecture is the best, but with gradual addition of code in small monolithic, it becomes a headache to handle everything and to understand each and every component of it. So, big monoliths have to be broken in small components called microservices for easy developement with increase in organization size.

IV. Architecture

Microservices are basically the decomposition of the business concerns in which each service handles it's need and responsibility independently or in loosely coupled fashion.

An example architecture Refer Figure 2 [4] By following this architectural style we ensure the following things[5]:

- Independent deployment
- Small team size for each service
- Highly scalable and maintainable
- Loosely coupled
- Better monitoring of services
- Language Independence
- Clear ownership
- Developer velocity
- System reliability

Microservices architecture is a method or a structure for developing various types of applications. It includes many small and autonomous services. Each service is separate in terms of code that can be handled by a compact development team.Communication in the services is done through well defined API's, where there is abstraction of structure among each other. It is not required or mandatory that the services share same framework and language.

API Gateway is the main component of microservices which act as an entry point for users. Users call the API Gateway rather than calling the services directly, which has its own importance as services can be re-engineered without requirement of any changes in all the clients. [15]

Microservices assists to build more supple and scalable applications. Besides monolithic applications which involved in a large scale development and quite a slow process, microservices can make the tasks more flexible and quick by structuring down a vast application into a cluster of microscale services which also promote independence in pace.Deployment of microservices is less challenging and even the failures are less cataclysmal than vast systems.It provides more reliability as when the number of users increase then the most accessed part of the application can be implemented using this architecture. [16]

The main challenge behind the implementation of microservices is decision about the pattern of decomposing the vast system into microservices.There are various schemes that can assist the structuring of that pattern:

- Characterise services related to business potential.
- On the basis of domain-driven design sub-domain.
- On the basis of use case and characteristics of the services that manage particular tasks e.g. delivery services that are in charge of delivery of finite orders.
- On the basis of resources by characterising a service that is in charge of all the activities on different resources. E.g. an account service that is in charge of the management of client accounts.

For the assurance of loose coupling, every service has its separate database. Application should use Saga Pattern [17] as a solution for the inability of two phase commit transactions in the maintenance of data integrity. A service issues an event when the data is modified. There are certain techniques for the modification of data and issuing events consisting of Transaction Log Tailing and Event Sourcing.

Queries for data retrieval that are governed by multiple services also set a challenge to be achieved. Command Query Responsibility Segregation(CQRS) [18] is the solution which is a pattern that under the read and update actions for the stored data. Performance,security and extendibility can be maximized by using CQRS [18]. The evolution of the system over a time period restrict the update operations from becoming the reason for merge issues at the defined level.

V. Limitations

We have found many limitations in the architecture as analyzed by us and that other's are also facing with this. Some of them we are going to discuss here.

A. *Interconnection delays in microservices*

- While developing any microservices based application, the main objective is to increase the efficiency of the application through proper integration of all microservices. Also, when we move from a monolithic application to any microservices based distributed design, the biggest challenge lies in forming an effective communication mechanism. The interconnection delay in microservices is a consequence of the communication between the internal services of the architecture. Here we need to get data from multiple microservices through API rather than getting it from a single application. Gradually, this makes an additional latency to the response time.
- We need to ensure an asynchronous communication (Refer Figure 3) for data propagation between microservices whenever possible. The focus is on preparing an autonomous microservice which serves even after the failure of other services that are a part of that application. Microservices having HTTP dependencies get their performances degraded in case any service in the request chain is not performing well which comes out to be a bottleneck.
- We use different protocols for client call and internal communication, even though both types are synchronous

[4]https://aws.amazon.com/microservices/
[5]https://microservices.io/

TABLE IV
CHALLENGES FACED WHEN BUILDING APPS WITH MICROSERVICES

Value	Percent	Responses
Monitoring	46.6%	292
Service Discovery	21.7%	136
App/Service stability	26.8%	168
App/Service security	27.2%	170
Deploying apps or services	25.4%	159
Changing culture to be open to microservices	38.2%	239
Others	5.1%	32
Not applicable	17.6%	110

TABLE V
CHALLENGES WHEN REFACTORING FROM LEGACY APPS TO A MICROSERVICES ARCHITECTURE

Value	Percent	Responses
Finding where to break up monolithic components	50.0%	313
Overcoming tight coupling	49.7%	311
Incorporating other technologies(containerization, etc)	23.6%	148
Testing	30.8%	193
Time investment	38.5%	241
Others	2.9%	18
Not applicable	21.2%	133

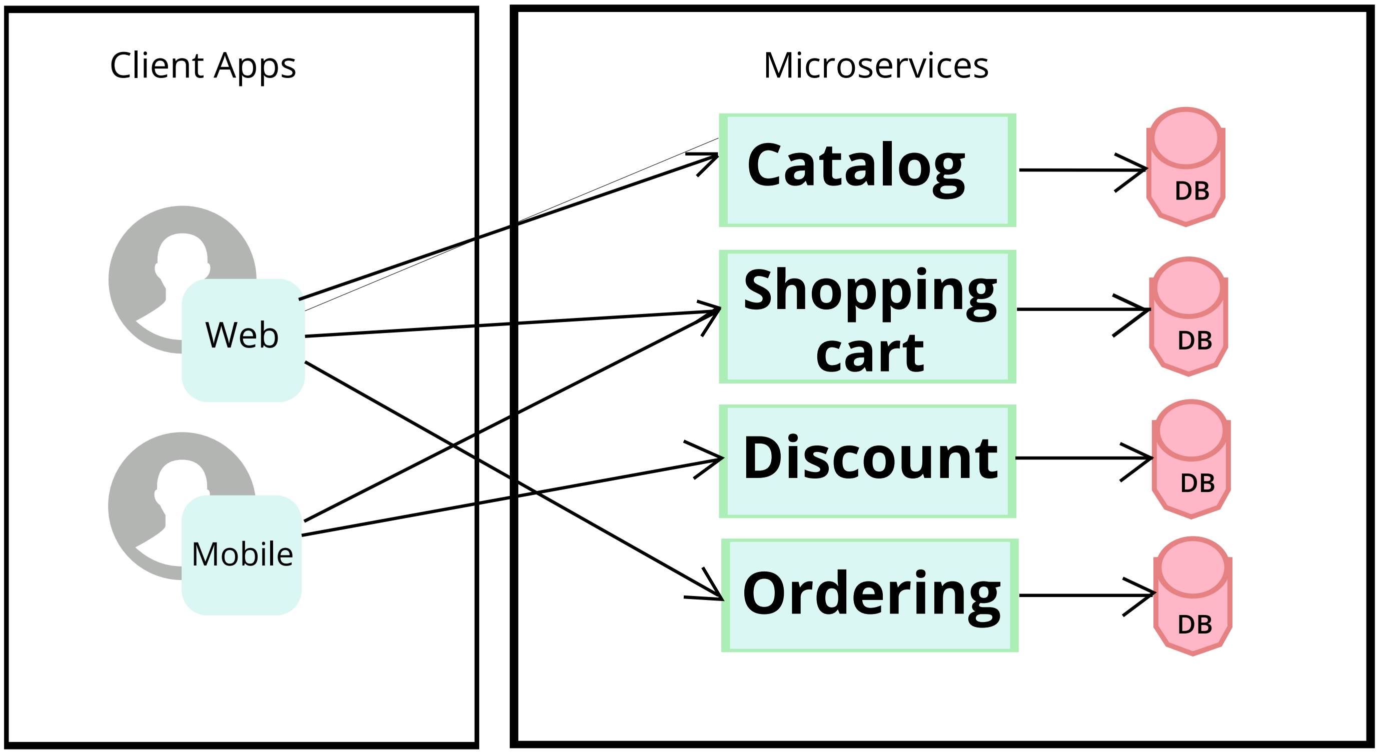

Fig. 1. Microservices workflow

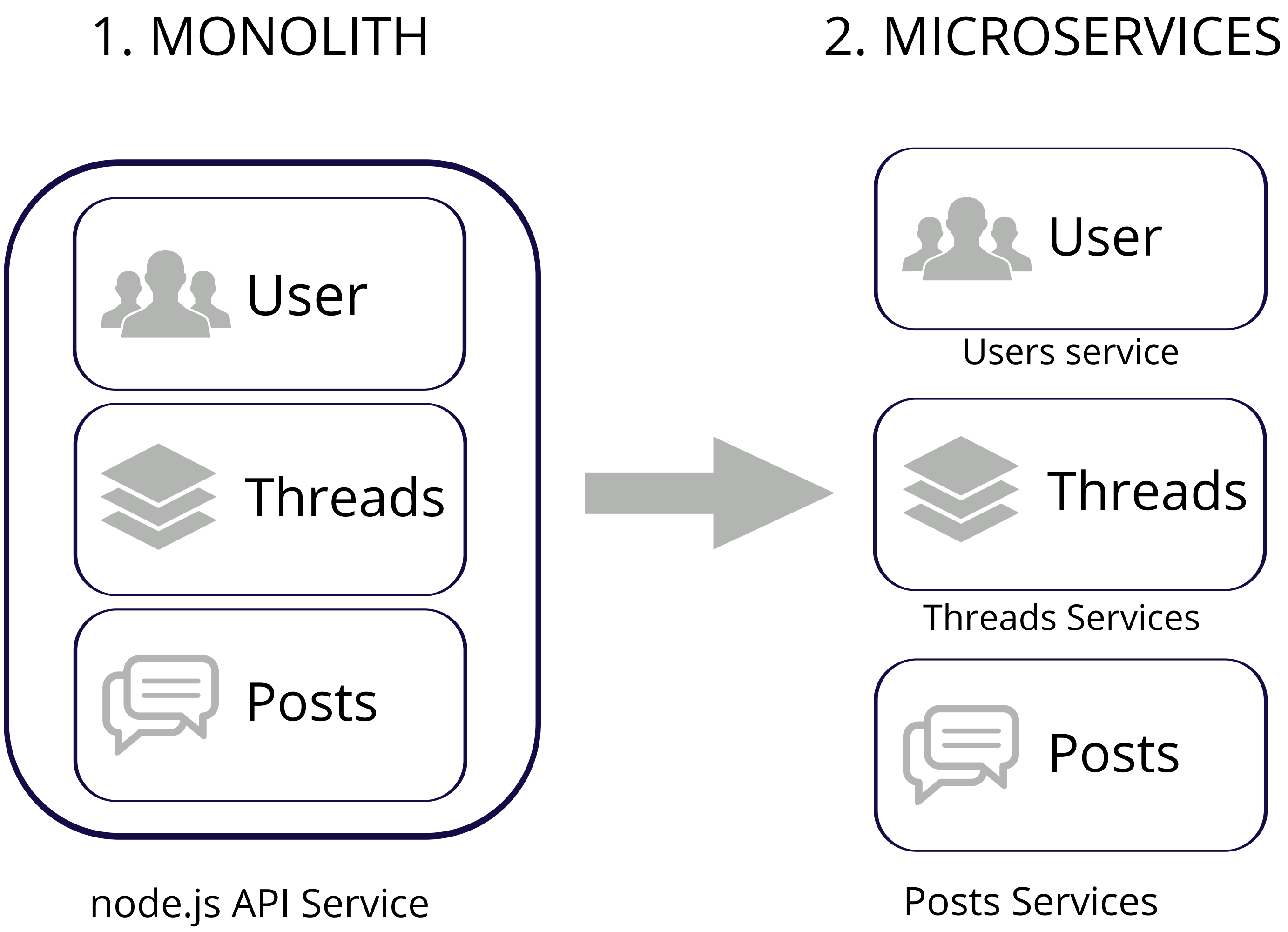

Fig. 2. Breaking a monolithic application into microservices

communication (Refer Figure 3). Because client requests are good to be REST in order to see payloads explicitly, back-end communication can be sacrificed to see payloads instead of picking velocity of response time. gRPC [19] is much faster than REST. So we can say that, if we prefer to communicate with synchronous communication, we have several options those are; HTTP protocols and REST approach. gRPC is binary format communications. [20]

B. Data lost and service discovery issues

- Failure is an uncertain part of any architecture we adapt in the whole process. But being resilient to those failures and to establish a fully functional state after a failure should be the ultimate goal. In comparison to any traditional monolithic application, the microservices architecture finds it really tough to maintain data consistency. There is always an ever-present risk of partial failure in these distributed systems. Data is vulnerable enough to get lost as the distributed system doesn't share any common memory.
- A monolithic codebase when split into smaller microservices based on problem domain and responsibilities, all the classes end up in different microservice projects. Therefore, methods of other classes cannot be called directly and REST API is the way to go forward. Unlike a monolithic architecture, smaller applications need to communicate with each other via REST API.
- So, how can one microservice know where the other one is deployed ? Hardcoding the URL in the microservice is not an optimal solution in case there are multiple instances that need load balancing. Service Discovery [21] is the solution for this which involves creating a new microservice called discovery server and it keeps track of where all the microservices are. It acts as a service registry [21] [22] who mediates the communication between various services. It consists of two models namely client-side service discovery [21] and server-side service [21] discovery.

C. Cost Overhead Issues

- One of the issues which highly hinders the adoption of microservices architecture is the increased adoption

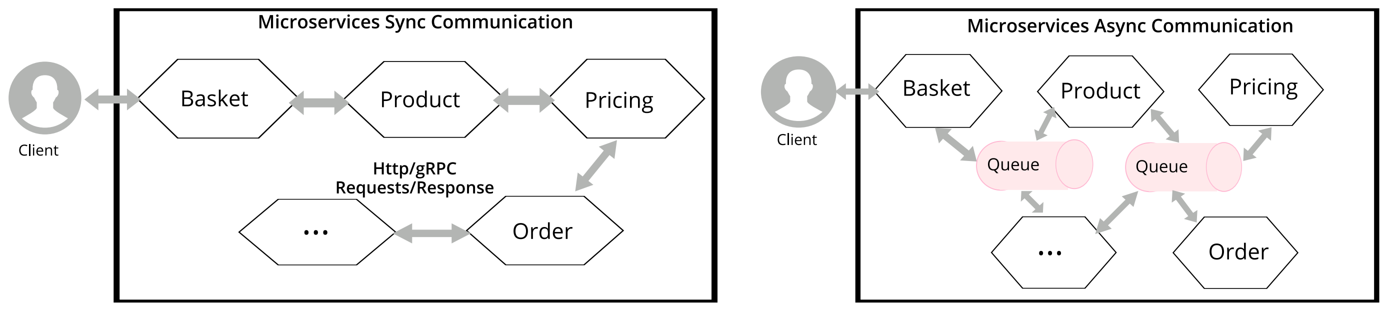

Fig. 3. Synchronous and Asynchronous Communication in microservices

cost of this architecture. It brings out a technological diversity to the whole process and allows various services to follow different technological bases. Hence, we might need to use different tools and techniques to perform same functionality according to the situation and the microservice. It is often recognized as a consequence of skill problems that arise when developers switch teams. Also, the maintenance of this architecture requires a big team to understand the diverse tool and technology landscape. This leads to an increased initial cost to manage skills and run time resources.

- Maintaining a more dispersed system requires a hidden cost for various areas such as a DevOps team, a SWAT team, duplicated cost of changes and common resourcing mistakes as well. But, it is believed that proper planning and careful resource management can help us minimize these costs. However, many researchers have estimated that in the long run of this architecture, this cost overhead can be highly compensated by the reduction of maintenance effort. [23]

D. Distributed Transaction management problems

- Microservices does not provide any management for transactions; we have to handle these with our algorithms. Normally a monolithic architecture is the one that has only a single database. But in distributed microservices, we work with eventual consistency. [24] Eventual consistency means accessing the data with low latency at the cost of using dirty data. It means that a monolithic architecture works with ACID property while distributed microservices work with BASE properties to achieve high availability. While ACID properties aim to achieve high consistency.
- Each microservice has its own databases which it uses to manage its internal transactions. For the execution of such a system, a perfect ordering of the internal operation of microservices is required. Successfully executing all the microservices implies that a transaction is completed. If there is an error in one microservice, then, all the microservices should rollback. It implies that all microservices roll back their operations in their databases.
- To achieve this goal, we can use a 2-phase commit protocol. In 2-phase commit protocol we have 2 phases: Prepare phase and the Commit phase. We have a Controlling site that leads and a Participating sites, i.e. microservices. In Prepare phase, all participating sites send a Prepare message. If a microservice is ready to commit, it sends a Prepare message 'Yes' otherwise 'No'. In the Commit phase, if all the prepare messages are 'Yes', then transaction commit. If any one of the prepare message is 'No', then rollback the transaction. But if the coordinator fails, the whole system will fail. The transaction will not commit until the slowest transaction among the participating sites is completed. Also, if one microservice faces a deadlock, then the other transaction has no option to rollback.

E. Increasing complexities in an ever-increasing microservice architecture

- We all know how rapidly technology is booming. Also, the functionalities that software provides are increasing day by day. With the advancement of functionalities in software, the complexity within the architecture of microservices is increasing. So our main aim, which was decomposing services into many independent microservices so there may be less load on each, is vanishing.
- Since the scalability of distributed systems is increasing, microservices architectures focus on decomposing the different components of software into independent domains. Monolithic architectures where the software has only one codebase has complex domain models. But microservices mainly focus on decomposing this one codebase into unique domains. It also helps in team building.

VI. Conclusion

Over the last few years, Microservice architecture has been gaining a significant popularity and is now a trending architecture. Some big enterprises such as Netflix and Spotify have successfully implemented this architecture and provide a factual motivation for other organizations to migrate to this technology. Independent deployability, being based on strong isolation, and easing the deployment and self management activities are the most widely adopted beneficial principles.

However, many companies are still hesitant to adopt this because they consider microservice as a avoidable hype or because they are not aware of the benefits and issues related to migration. Also, there exist no single microservices pattern that is better than the others. Rather, it all depends on the scenario and needs of any organization to shift to a new architecture. These architectures increase productivity and deflate costs as they are highly efficient in the long term.

This paper presents an empirical study and research related to the microservices architecture and highlight the limitations that hinder its migration and growth. Microservices architecture has already reached a certain degree of maturity in context to industrial state-of-practice. Still, there exist some sort of gap between academic state-of-art and industrial state-of-practice. Here, we have successfully complemented its literature on gains and pains of microservices. Practitioners can hence exploit our findings for day-by-day work with microservices. This paper aims to add to the literature by identifying and presenting a range of issues linked with microservices.

In the scope of our future work, we plan to extend our analysis by studying this architecture from industry perspective and devise a methodology which meters the performance downfall caused by the limitations we have expressed here. We also plan to identify the best practices of this architecture which are critical to it's successful incorporation in the future of SOA (Service-Oriented Architecture).

References

[1] K. Gos and W. Zabierowski, "The comparison of microservice and monolithic architecture," in *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. IEEE, 2020, pp. 150–153.

[2] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017.

[3] B. Kitchenham, P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering-a systematic literature review," *Information and Software Technology*, vol. 51, pp. 7–15, 01 2009.

[4] V. Dwivedi, V. Pattanaik, V. Deval, A. Dixit, A. Norta, and D. Draheim, "Legally enforceable smart-contract languages: A systematic literature review," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–34, 2021.

[5] S. Baškarada, V. Nguyen, and A. Koronios, "Architecting microservices: Practical opportunities and challenges," *Journal of Computer Information Systems*, 2018.

[6] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, vol. 150, pp. 77–97, 2019.

[7] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study." in *CLOSER*, 2018, pp. 221–232.

[8] M. Kalske, N. Mäkitalo, and T. Mikkonen, "Challenges when moving from monolith to microservice architecture," in *International Conference on Web Engineering*. Springer, 2017, pp. 32–47.

[9] J. Ghofrani and D. Lübke, "Challenges of microservices architecture: A survey on the state of the practice." *ZEUS*, vol. 2018, pp. 1–8, 2018.

[10] H. Zhang, S. Li, Z. Jia, C. Zhong, and C. Zhang, "Microservice architecture in reality: An industrial inquiry," in *2019 IEEE international conference on software architecture (ICSA)*. IEEE, 2019, pp. 51–60.

[11] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri, and T. Lynn, "Microservices migration patterns," *Software: Practice and Experience*, vol. 48, no. 11, pp. 2019–2042, 2018.

[12] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How to make your application scale," in *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*. Springer, 2017, pp. 95–104.

[13] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Microservices in industry: insights into technologies, characteristics, and software quality," in *2019 IEEE international conference on software architecture companion (ICSA-C)*. IEEE, 2019, pp. 187–195.

[14] J. Vučković, "You are not netflix," in *Microservices*. Springer, 2020, pp. 333–346.

[15] J. Zhao, S. Jing, and L. Jiang, "Management of api gateway based on micro-service architecture," in *Journal of Physics: Conference Series*, vol. 1087, no. 3. IOP Publishing, 2018, p. 032032.

[16] E. Price, S. Wray, A. Buck, D. Lee, and D. Coulter, "Microservices architecture style," https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices, [Online; accessed 27-January-2022].

[17] C. Richardson, "Pattern: Microservice Architecture," https://microservices.io/patterns/microservices.html, 2018, [Online; accessed 27-January-2022].

[18] E. Price, A. Buck, D. Kshirsagar, C. Gronlund, and U. Dahan, "CQRS pattern," https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs, 2021, [Online; accessed 27-January-2022].

[19] X. Wang, H. Zhao, and J. Zhu, "Grpc: A communication cooperation mechanism in distributed systems," *ACM SIGOPS Operating Systems Review*, vol. 27, no. 3, pp. 75–86, 1993.

[20] B. Christudas, "Microservice performance," in *Practical Microservices Architectural Patterns*. Springer, 2019, pp. 279–314.

[21] S. Gadge and V. Kotwani, "Microservice architecture: Api gateway considerations," *GlobalLogic Organisations, Aug-2017*, 2018.

[22] C. Richardson, "Pattern: Service registry," https://microservices.io/patterns/service-registry.html, 2018, [Online; accessed 27-January-2022].

[23] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.

[24] S. Newman, *Building microservices*. O'Reilly Media, Inc., 2021.