DESIGNING SCALABLE AND MAINTAINABLE APPLICATION PROGRAMS

Swamy Prasadarao Velaga
Software Engineer, Department of Information Technology

ABSTRACT

This paper aims to discuss the following on how MSA has revolutionized the manner of developing application programs: This literature review shows how microservices have evolved from the monolithic applications and Service-Oriented Architecture (SOA) models with Microsoft decomposing monolithic architectures with the launch of MSA. The switch to applying microservices, on the same hand, holds a number of advantages, such as the possibility to scale and modularize, in addition to the series of problems to solve with a deeper strategic vision.Microservice architecture is a way of designing applications as a collection of loosely coupled services. It is an architectural style that structures the application as a set of autonomous services, which are modeled around a business domain. When building an application as a set of microservices, the primary goal is to keep each service as single-minded as possible[1]. Each service can be developed, tested, and deployed independently and has its own process and can use a different technology than the other services in the same application. The design of microservices requires a more mature approach to software development as the degree of complexity shifts from intra-service cooperation to the landscape of service cooperation. The complexity of the communication between services is at the core of this architectural style, and despite many benefits, microservices have a downside, as they introduce some level of complexity, especially in the area of deployment and figuring out the service inter-communication and data consistency when data is being held by different services. Microservices development and maintenance presents different issues to the architecture such as growth in the number of services, tangled dependencies and versioning. As a result of these maintenance challenges some of the solutions that have been described in this review include automation and management tools. It is necessary to integrate it with the DevOps practices, most critically with Continuous Integration and Continuous Deployment (CI/CD) practices to help the general functioning of microservices [1]. This paper aims at analyzing the use of microservices together with DevOps, examination of the roles of their cooperation in development of efficient deployment processes. It may be able to notice that security in MSA is a very important factor, some of the problems may include, service authentication, data integrity and inter service communication security. The following is a review of these security threats and measures that can be taken to avoid them to achieve strong security protection in Microservices platforms [1,2]. Microservices need to be scalable and deceptively easy to deploy and containerization tools like Docker and container orchestration tools like kubernetes fit this need perfectly. The following paper focuses on their effectiveness within the operational environment as well as system stability. Although microservices architecture provides benefits, monitoring and observability have to be considered so that microservices' issues will be detected and solved in due time. It studies measures for increasing understanding of the application's characteristics, enabling timely further action in case of problems. Finally, the paper explores some of the design patterns that address service vulnerability, and the following are the common ones; circuit breaker design pattern, service discovery pattern, and the load balancer design pattern. In striving for achieving this objective, the review seeks to offer a vast amount of information regarding the various concepts that help to lay a solid foundation that can enable the creation of a well-scaled microservices-based application.

E-ISSN NO: 2349-0721

Keywords— ***Microservices Architecture, Scalability, Maintainability, Monolithic Architecture, Service-Oriented Architecture (SOA), Best Practices, Design Patterns, Anti-Patterns, DevOps, Continuous Integration (CI), Continuous Deployment (CD), Security, Containerization, Orchestration, Monitoring and Observability***

INTRODUCTION

Microservices Architecture or MSA has become an important concept in the last couple of years in the field of software engineering and is considered as a new architectural style of software application development. When compared with the single-tier application, a microservices architecture splits an application into multiple, related but self-contained elements. Each sub-service is a singular service that deals with a particular capability in the business and it interacts with other sub-services through a contractual interface referred to as API [3]. This architectural style encourages solutions to be built with scalability in mind and be very maintainable as well as able to quickly respond to change requests, which makes it ideal for today's complex web applications, inherently built for cloud infrastructure. Microservices have developed over time and can be attributed to the increasing difficulties and constraints characteristic of monolithic applications. Monolithic solutions, in particular, face problems with scalability since the update of any application means redeployment of all parts of code. Also, keeping a vast monolithic application becomes difficult and brings issues because one part's integration impacts others. Microservices, on the other hand are a more subdivided way of development and deployment, whereby modish teams can build, implement, and expand selected services of the application without impacting the other components [3].

Microservices are smaller individual components which are loosely bounded and services oriented where each component can deliver a single business capability and is accessed via an API. It is seen that the process of implementing business capability in microservices is ideal for scaling development; one team can focus on a specific business capability and improve it separately from others. Microservices also can be implemented with various technologies and also can change over time more than an enterprise monolithic application which tends to hinder enterprises that attempt to apply some form of continuous development and delivery. In the last few years, many organisations have embarked on this process to decompose an enterprise monolith into fragments. Nevertheless, as it is often the case with most architectural styles, the microservices style is not without its difficulties. Throughout this chapter, the reader will learn about some routine and rather unusual tips and issues connected with the application of microservices architecture and its functioning [4]. The following attests to the fact that there exist various benefits to enterprises upon the adoption of the microservices architecture. That is why many specialists use the Strategy of Decomposing Business Capability to independent small components that evolve separately from each other within a development team. Also, the opportunity to utilize various technologies whereas the scale of development can be changed separately also contribute to the CCDC process [5]. However, the prescriptions come with their fair share of problems that need to be assimilated as well. With that insight, organizations can maximize the possibilities of a microservices architecture while minimizing the possibilities of risks. Applications are being developed today in microservices architecture which is known for its flexibility and scalability. This architectural pattern enables organizations to construct and implement numerous diminutive services that can be individually managed, thus improving the system's velocity and development. Nevertheless, Just like any other architectural transformation the shift to microservices has its own set of problems. Having and running many, separate, standalone services may seem as a good idea and practice but in fact it may be very costly and inefficient [6]. Furthermore, numerous changes are expected during the process of transitioning from a monolithic architecture to the microservices one, including the cultural change within an organization. Although

E-ISSN NO: 2349-0721

the aforementioned challenges are cumbersome, companies can easily overcome them with the right strategies and best practices in order to have the best of a microservices architecture.

RESEARCH PROBLEM

The main research problem in this study is to identify the best practices and challenges of designing microservices architecture (MSA) that may affect its implementation. Despite the recent increasing interest in MSA and the available best practices and guidelines, there is not enough in the literature discussing detailed descriptions about the potential problems that developers may face and how to address these problems. Consequently, developers may face difficulties transferring the existing knowledge into practical use when implementing MSA. Therefore, to help developers and architects of MSA, we report our experience and the lessons learned from building seven real-world MSA systems over a period of four years. Based on our practical experience, we describe the challenges and articulate a set of best practices in terms of these challenges. The identified challenges and best practices are then validated using a perception survey during a workshop [7].

Microservices architecture has gained a recent fame in enterprise software development due to its several attractive characteristics. However, implementing software systems using microservices presents a set of unaddressed new challenges, in addition to other well-known challenges in distributed systems [7,8]. These sets of challenges could hinder the successful deployment of microservices architecture. In the literature, despite the available guidelines and best practices of how to design and implement microservices-based systems, there is not enough detailed description about the potential problems that developers may face and how to address these problems [8]. We believe that an ultimate understanding of challenges is necessary to modify the existing best practices and guidelines into practical rules. Therefore, in this paper, we clearly articulate a set of implementation challenges found while building real-world microservices systems, and we detail a set of best practices in terms of these challenges—all based on our practical experience.

LITERATURE REVIEW

A. MICROSERVICES ARCHITECTURE

Microservices architecture has gained increasing popularity in modern software development. Large companies such as Netflix, eBay, and Amazon have adopted this architecture and have shown significant benefit from it. Microservices are small, independent, and loosely coupled services. These services are modular and each of them describes a specific business function that can be implemented independently of the others. However, the usage and interpretation of the microservices architecture is different to some extent. Some just use it as a title while others have moved to using it as a title to a professional level. In this chapter, what has been stipulated is the current approach to the design and implementation of microservices [8]. It will describe and showcase the several research's key findings on best practices and challenges concerning the several areas pertaining to microservices' implementation. Moreover, it will discuss the microservices' architecture and the current frameworks that are in place to accommodate microservices as well as the available cloud platforms and their support to microservices. As the need for the architecture that is more scalable and adaptable rises, microservices have rightfully gained the status of one of the main approaches for developing software in the contemporary world [8]. One of the advantages of microservices is that the developers have been able to design and develop complex and scalable software systems because of the flexibility and modularity of the architecture. Moreover, the development of cloud computing also promotes microservices more because it offers a solid and inexpensive environment for implementing the microservices-based applications. Thus, in the course of the further development of digital transformation, the requirement for reliable and highly efficient microservices architectures will remain relevant

E-ISSN NO: 2349-0721

and only increase. Therefore, the developers and architects working on microservices architecture need to be fully aware of the major concepts and recommended approaches to the creation of the reliable microservices architecture systems.

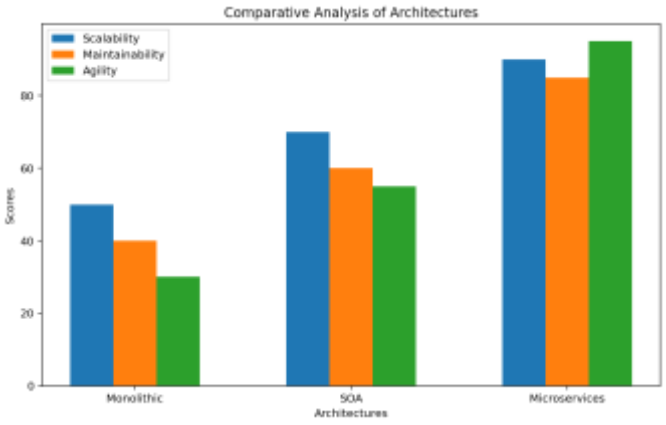

Fig. 1 Comparative Analysis of Architectures

A. BEST PRACTICES: PATTERNS AND ANTI-PATTERNS

When developing a system, experienced architects will use both patterns and antipatterns. Our collective experience has shown that understanding these potential pitfalls is paramount for success in the development and maintenance of microservices. By avoiding these common mistakes and misconceptions, teams can work toward a more efficient and effective microservices architecture, ultimately leading to better outcomes for both developers and end users.

We first introduce key patterns to support microservices architecture and then discuss various microservices antipatterns as well. Microservices architecture is underpinned by a myriad of patterns aimed at ensuring their efficient and effective operation. These include an array of patterns that are instrumental in structuring and managing the microservices themselves [9]. Such patterns encompass the API gateway, backing service, circuit breaker, externalized configuration, logging and health check patterns, service discovery, and self-contained services. We believe that the conventional service-oriented architecture (SOA) patterns are not succinct enough for modern architects, and therefore we have meticulously assimilated and elucidated a range of microservices-specific patterns. These are supplemented by other patterns that focus on the intricate interactions between microservices, such as database per service, event sourcing, external event publisher, and competing consumers patterns, in addition to a multitude of messaging-related patterns that facilitate asynchronous communication between microservices [10].

B. INTEGRATING MICROSERVICES WITH DEVOPS

DevOps is not just about developers and the IT operations team getting together to have a conversation. DevOps is about developers, testers, and experts from IT operations coming together in order to produce flow within IT and to increase deployment frequency, which can lead to higher quality and better security as well as more awareness of and better collaboration around business objectives. Integrating microservices with DevOps will be significant. Microservices, when related to DevOps, will be more than a development practice or an architecture. When integrated with DevOps, microservices can deliver the best business results [10]. The approach, the essence of DevOps, lies in using a unified software development and infrastructure automation practice with cultural philosophy to enhance collaboration among software developers, quality assurance, and IT operations.

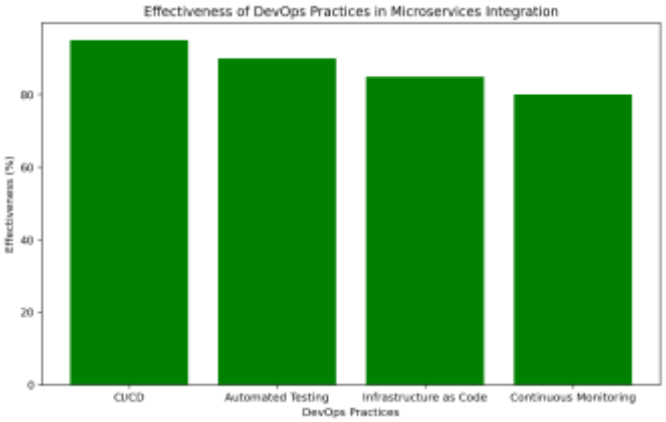

Fig. 2 Effectiveness of DevOps Practices in Microservices Integration

The combination of microservices and DevOps practices allows software development to keep pace with businesses' increasing demands for new features and capabilities [11]. DevOps solves that by automating the process of building, deploying, and operating software, so Dev and Ops teams can take a collaborative approach to managing software systems. DevOps ultimately solves the problem of speed at the organization level. DevOps promotes a cultural shift, a renewed focus on automation, a new level of tool integration, and a more dynamic way of scaling to support fast-moving software processes. Microservices architecture when integrated with DevOps best practices can accelerate the development and delivery of complex applications, and at the same time improve its robustness. DevOps is making the process of delivering software more efficient on the business level.

A. Role of Containerization and Orchestration

Microservices are small, independent, and loosely coupled services that follow the Single Responsibility principle and are organized around specific business goals. Microservices architecture is fast, reliable, separate, and scalable. It is easy to define, develop, deploy, and run the microservices. Super Domain is divided into multiple Micro Domains. Each Micro Domain contains a business subdomain and related business capabilities. Each business capability has one or more microservices. These microservices access related business data and encapsulate related business rules [12].

To efficiently and effectively develop and deliver the microservices-based software applications, the best practices as discussed earlier need to be followed. In addition, there are some challenges related to the security, testing, and DevOps that need to be addressed to incorporate microservices successfully in the software development and deployment. Containerization and Orchestration of microservices help to address some of these challenges in the implementation and deployment of the Microservices architecture. There are various containerization technologies available to package the microservices [12]. Docker is most commonly used as a containerization technology. With Docker, it becomes easy to package each microservice with all the dependencies and libraries into a container. Then, the containers can be shipped into any environment and executed. Hence, Docker containerization helps to reduce the "it works on my machine but not in the production" problem that happens during the deployment of microservices.

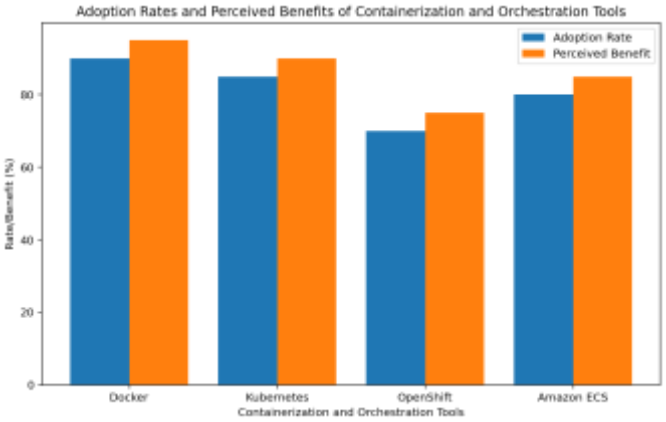

Fig. 3 Adoption Rates and Perceived Benefits of Containerization and Orchestration Tools

B. MONITORING AND OBSERVABILITY TOOLS

Commonly used tools include DataDog, NewRelic, AWS CloudWatch, Azure Monitor, GCP Stackdriver, Graphite, Sensu, InfluxData, Nagios, Prometheus, ELK (Elasticsearch, Logstash, Kibana), Splunk, Fluentd, Sysdig, Zipkin, Jaeger, LightStep, Dapper, and others. Demand and requirements constantly fluctuate, making observability essential for understanding the data, workflow, and dependencies of the change. The best practices for implementing microservices address key issues related to team structure, committing code to a version-controlled code repository, defining small services, using a container to deploy microservices, deploying the same microservices to multiple environments, scaling services automatically, creating each service with its own data storage, building a resilient system, incorporating monitoring during the development pipeline, and running continuous testing to help improve delivery speed and quality [12].

While building and running applications on private or public cloud environments, it is essential to have a good understanding of what is happening at every layer in the application and infrastructure. Traditional tools such as snoop, strace, log, etc. may only work in a small company environment, but they become inadequate in the cloud environment. Moreover, the cloud environment makes it difficult to debug and is more error-prone. To improve reliability in the microservices architecture, the implementation best practice is to include monitoring in the development pipeline. APIs can be tested, load conditions can be evaluated, and error conditions can be developed and tested before deploying the microservice [13,14]. Microservices are deployed individually, which allows for individual services to be tested under frontend and backend conditions before being marked as ready for production. Tools for monitoring can be classified as agent-based tools such as DataDog, NewRelic; agentless such as Prometheus, sysdig; and service mesh such as Linkerd, Istio, Kuma, Consul.

II. CONTRIBUTIONS

My contribution in this study is to integrate knowledge and information available concerning microservice architecture and the key design principles and practices for constructing the application programs that are scalable and maintainable. This review synthesizes the various scattered literature reviews, theoretical analysis and findings of the academic researches, industry reports and the technical journals which focus on the general area of the review. contribution #1 proposes the aggregation of the different best practices to define an architecture for microservices. Thus, summarizing the information from various sources, I reveal the fundamental tendencies and best practices that can contribute to the proper scaling of microservices-based systems and their further sustainable development. Another strength of my study is the elucidation of objectives, definition and analysis of concerns

E-ISSN NO: 2349-0721

related to ensuring microservices based application and proffering of solutions and strategies in relation to these concerns. Thus, by identifying and analysing challenges related to services and dependencies, such as service proliferation, numerous and intertwined dependences, and security threats, My study provides concrete suggestions on how to minimise these issues. My work seeks to augment the existing knowledge on the utilization of microservices in supporting architecture and more specifically on how DevOps, especially the CI/CD process, can be implemented. In the context of defining the relation between microservices and DevOps, I elaborate how automation, collaboration, and adaptability are significant in contemporary software development.

In addition, my research enriches the advancement of the new technologies which are related to the containerization and orchestration in Microservices architecture. In this paper, I discuss the effect these technologies such as Docker and Kubernetes have on the microservices' architecture, and the way they facilitate the set architecture's scalability and relevance by deploying the microservice. Therefore, my study provides directional and procedural advice for architects, developers, and organizations who wish to adopt Microservices architecture. Based on the literature, there is rich guidance on designing, implementing, and supporting microservices-based applications in practice that I synthesize as recommendations.

III. SIGNIFICANCE AND BENEFITS

Enterprise information systems are crucial for any size of organization to support and help in making complex business decisions and gain competitive advantage. These enterprise systems have traditionally been deployed as large monolithic artifacts [14,15]. The size, scale, and complexity of the corresponding software applications have managed to grow to gigantic proportions. Though monolithic applications are easy to develop and deploy initially, over a period of time they become very large and complex and are extremely difficult to scale. The web era of such enterprise applications saw a rapid increase in the deployment of monolithic applications with the advent of J2EE and .NET platforms. Such monolithic applications are typically made up of multiple layers, such as presentation, business, and data, which form an indivisible unit and are deployed as one application [16]. Organizations, including social and e-commerce platforms, are trying to react to business challenges with software application development and deployment. Rapid changes in software development and deployment occur when seasoned architects employ new approaches with the design and implementation of software applications. In recent times, there has been a lot of hype surrounding microservices architecture about how this style of architecture can provide solutions to most of the problems associated with monolithic applications that have surfaced over the years [17,18].

Microservices as one of the patterns in software application architecture is rapidly growing and is one of the most famous topics in modern enterprise software development and deployment. That is not to say the microservices architecture is not a silver bullet and indeed, it is a topic of much contention as to which style of architecture is best suited to the organization's needs. This is so because regardless of the architecture style in the development of software applications basic issues crop up in the lifecycle of the application that need to be addressed. The mastery of these issues proves a core and dominant factor towards success or failure of a software application [19]. Hence, organizations should cautiously reflect on their needs and select the suitable architecture style that fits the organizations' standards, mission, and vision. To this end, one must understand that the decision of choosing proper software architecture is another critical factor that defines organizational flexibility within the constantly evolving business context. Enterprise information systems: anatomy of architecture In today's fast and

dynamic world, organizations need to understand and analyze the complexity of a business environment comprehensively.

Therefore, architecture should be a significant concern for any organization. One of the major reasons behind the decision of large monolithic systems, many a time even replacing prior solutions, is that it actually reduces the general complexities concerning the development and the deployment practices. But at the same time, it can have issues concerning flexibility and scalability if applied in a large-scale system. Incorporate all the information accessible to you related to microservice architecture and the main principles and recommendations for building the application programs which are scalable and easily maintained[19]. By and large, this review consolidates the several fragmented literature review, theoretical discussion and conclusion of the related academic researches, commercial studies and technical journals which are concentrated in the wider field of the review. contribution #1 is composed of the following three steps: aggregation of approximately twenty best practices about microservices to form a list of characteristics; the elaboration of an architecture for microservices based on the list of characteristics above; and the fit/gap analysis to check the suitability of the selected characteristics for the five use cases we have presented in the research. Therefore, briefly sharing and comparing the data obtained from different sources, I define the general trends and effective strategies that can help maintain the accurate growth of the microservices-based systems and their further evolutionary advancement.

CONCLUSION

The main aim of this paper was to perform a thorough analysis of microservices architecture with the emphasis placed on the use of the architectural model to design extensible and manageable application programs. Thus, focusing on conceptual research on the topic, this paper aimed at identifying the major tenets, recommendations, issues, and trends in microservices architecture. Furthermore, the paper aimed at offering practical guidelines and suggestions to the architect, developers and organization who plans for using the microservices for developing the modern software solution. It was important to identify the interaction of microservices architecture with DevOps methodologies and focus on automation, collaboration, and continuous delivery as the keys to successful microservices implementation. The paper also sought to analyze the use of such new technologies like containerization and orchestration to boost the flexibility and performance of systems based on microservices.he design and implementation of business support systems is now being presented as a service-oriented architecture that encapsulates the business rules and processes within the complex hardware and software infrastructure. The best practices for the microservices architecture emphasize modularity, the single responsibility principle, a preference for private overprotected, and the avoidance of public constants. These principles serve as a guide for the development and implementation of modern software systems. This research was prompted by the increasing interest in microservices within the industry and the lack of academic studies on the microservices architecture. We sought to investigate the primary challenges that companies encounter when adopting the microservices architecture style. The findings revealed that the most significant challenges revolved around maintaining data consistency, which was cited as challenging by 48.4% of the participants. Following closely was the difficulty in establishing the appropriate service boundaries, reported as challenging by 39.7% of the participants. Furthermore, coping with the heightened inter-service communication proved to be a notable challenge for 36.6% of the participants. Additionally, our study identified a positive correlation between company size and the frequency of challenges experienced during microservices implementations. Notably, the analysis of subgroups did not reveal significant disparities in perceived complexities between more experienced and less experienced teams in the realm of microservices.

E-ISSN NO: 2349-0721

REFERENCES

[1] R. Daigneau, *Service design patterns : fundamental design solutions for SOAP/WSDL and RESTful web services*. Upper Saddle River, Nj: Addison-Wesley, 2012.

[2] A. B. Bondi, "Characteristics of scalability and their impact on performance," Proceedings of the second international workshop on Software and performance - WOSP '00, 2000, doi: https://doi.org/10.1145/350391.350432. Available: https://dl.acm.org/citation.cfm?doid=350391.350432.

[3] P. Smeys, P. B. Griffin, Z. U. Rek, I. D. Wolf, and K. C. Saraswat, "Influence of process-induced stress on device characteristics and its impact on scaled device performance," IEEE Transactions on Electron Devices, vol. 46, no. 6, pp. 1245–1252, Jun. 1999, doi: https://doi.org/10.1109/16.766893

[4] A. Erramilli, O. Narayan, and W. Willinger, "Experimental queueing analysis with long-range dependent packet traffic," IEEE/ACM Transactions on Networking, vol. 4, no. 2, pp. 209–223, Apr. 1996, doi: https://doi.org/10.1109/90.491008

[5] P. Kuchana, Software architecture design patterns in Java. Boca Raton, Fl: Auerbach Publications, 2004.

[6] M. Silver, Programming cultures : art and architecture in the age of software. London: Wiley-Academy, 2006.

[7] A. W. Scheer, Business process excellence : ARIS in practice. Berlin ; New York: Springer, 2002.

[8] D. Giannakopoulou and F. Orejas, Fundamental Approaches to Software Engineering. Springer, 2011.

[9] P. Jogalekar and M. Woodside, "Evaluating the scalability of distributed systems," IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 6, pp. 589–603, Jun. 2000, doi: https://doi.org/10.1109/71.862209

[10] C. M. Woodside and C. Schramm, "Scalability and performance experiments using synthetic distributed server systems," Distributed Systems Engineering, vol. 3, no. 1, pp. 2–8, Mar. 1996, doi: https://doi.org/10.1088/0967-1846/3/1/002

[11] X. D. Zhang, Y. Yan, and K. Q. He, "Latency Metric: An Experimental Method for Measuring and Evaluating Parallel Program and Architecture Scalability," Journal of Parallel and Distributed Computing, vol. 22, no. 3, pp. 392–410, Sep. 1994, doi: https://doi.org/10.1006/jpdc.1994.1100

[12] X.-H. Sun, "Scalability versus Execution Time in Scalable Systems," Journal of Parallel and Distributed Computing, vol. 62, no. 2, pp. 173–192, Feb. 2002, doi: https://doi.org/10.1006/jpdc.2001.1773

[13] X. Hu, L. Liu, and T. Yu, "A hierarchical architecture for improving scalability and consistency in CVE systems," International Journal of Parallel, Emergent and Distributed Systems, vol. 26, no. 3, pp. 179–205, Jun. 2011, doi: https://doi.org/10.1080/17445760.2010.495722

[14] A. Almojel, "Evaluating Budget of Overhead and Scalability on High-Performance Computing Systems," Journal of King Abdulaziz University-Science, vol. 17, no. 1, pp. 207–222, 2005, doi: https://doi.org/10.4197/sci.17-1.20

[15] J. L. Bosque, O. D. Robles, P. Toharia, and L. Pastor, "Evaluating scalability in heterogeneous systems," The Journal of Supercomputing, vol. 58, no. 3, pp. 367–375, Mar. 2011, doi: https://doi.org/10.1007/s11227-011-0593-5.

[16] M. Petticrew and H. Roberts, Systematic Reviews in the Social Sciences. John Wiley & Sons, 2006.

[17] G. Barish, Building scalable and high-performance Java Web applications using J2EE technology. Boston: Addison-Wesley, 2002.

[18] R. Redler and J. Rossberg, Designing Scalable .NET Applications. Apress, 2008.

[19] P. Runeson, M. Host, A. Rainer, and B. Regnell, Case Study Research in Software Engineering. John Wiley & Sons, 2012.