Evaluating the Inter-Service Communication on Microservice Architecture

L.D.S.B Weerasinghe
Department of Computer Science & Engineering
University of Moratuwa
Sri Lanka
weerasingheldsb.20@uom.lk

I Perera
Department of Computer Science & Engineering
University of Moratuwa
Sri Lanka
indika@cse.mrt.ac.lk

2022 7th International Conference on Information Technology Research (ICITR) | 979-8-3503-3203-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICITR57877.2022.9992918

***Abstract*—Distributed computing concepts have proliferated with cloud computing concepts in the past decade. With the evolution in cloud computing, Microservice architecture has significantly become popular as a new architectural pattern and a software development architecture. Most enterprise software development has moved their monolithic software architecture to microservice-based architecture, as it can divide large applications into light-weighted, distributed components. However, this approach may be subject to certain downsides as well. With the modern convention, engineers have succeeded in achieving scalability and maintainability quality attributes and lack the performance attribute in terms of response time. This is because the microservice architecture has introduced inter-service communication over the network. The key challenge when developing a microservice-based application is choosing the correct inter-service communication mechanism to reduce the time taken when calling each service. This research has taken an experimental methodology to compare and contrast the most trending inter-service communication mechanisms. Industry-standard benchmark load test is run to collect quantitative data to evaluate the overall system performance in terms of response time. The testing observed that gRPC protocol performs well in terms of response time and throughput compared to the HTTP and Web Socket protocols.**

Keywords—Microservices, Inter-service communication, Performance, Latency

I. INTRODUCTION

Cloud computing has presently become the most popular technology in the world. Hence, people have moved their infrastructure from server farms to the cloud. Cloud providers offer Infrastructure as Service (IaaS), Platform as a Service (PaaS), Software as a service(SaaS), and Function as a Service(FaaS). On the other hand, managing on-premise servers has many drawbacks, such as high maintenance costs, infrastructure provisioning and troubleshooting taking more time, limited scalability, etc. When moving to cloud-based solutions, people can achieve the below advantage,

- Better and quick scalability
- No infrastructure maintenance cost
- Lower energy cost
- Lower risk of unavailability
- Can access it from anywhere at any time

When the industry is seeking cloud-native solutions for their enterprise application, they must make that Software cloud-native friendly. However, bulky Software like monolithic systems is not precisely cloud-native friendly as they are cumbersome and need high-end computers for running such monolithic applications. With the separation of the concept, people are looking at Service Oriented Architecture(SOA). The main drawback of SOA is that it uses the Enterprise Service Bus(ESB), which orchestrates all the services [1]. Therefore, overall system performance is getting degraded.

Over the past decade, microservice architecture has become immensely popular among popular industries such as Netflix, Amazon, etc. They attain to serve millions of transactions per second using microservice architecture. There are many underlying reasons behind microservice architecture becoming popular in the software industry. A few of them are as below,

- Independently scalable. (Can scale necessary component only)
- Ease of understanding the code base
- Better maintainability
- Easy fault isolation
- Eliminate vendor or technology lock-in

The main concept behind the microservice architecture is that the business functionalities become smaller independent services. As human requirements become complex day by day, the architects are resolved to map those requirements to the Software to get the needed tasks done concisely and straightforwardly. Since the Software frequently gets many change requests, microservice architecture can facilitate such requirement changes. There are a lot of microservices frameworks that help to acquire frequent modifications to the microservices and other quality attributes such as scalability, security, etc. In order to achieve those quality attributes, different programming languages provide several microservice frameworks, such as Java provides Springboot and Vert.X, Go language provides the GoMicro framework, Node Js language provides the Molecular framework, and so on.

However, even while developing the microservices with these frameworks, software developers face certain challenges as follows,

- There is no standard way to break down massive monolithic systems into microservices.
- Inter-service communication between microservices.

979-8-3503-3203-2/22/$31.00 ©2022 IEEE

- Managing the transactional database queries in the distributed environment by protecting the ACID properties.
- Service discovery when service scales up and scales down.

This paper focuses on the inter-service communication between the microservice in the distributed environment. This is the most challenging and vital area concerning microservice architecture performance. In the monolithic system, all communications are conducted at the programming language level and byte code-based communication. Hence, there is no latency added when exchanging data. When it comes to microservice architecture, inter-service communication stays at a significant place because it is involved with network layer data exchange.

II. STATE OF THE ART

In the last decade, microservice architecture has become the most predominant and adopted software architecture in the service-oriented industry [2]. Services are built from the major business functionalities and deployed as independent services in a distributed environment. From that, people can achieve scalability, resilience, loose coupling, maintainability and so on [3]. Baylor University conducted a case study regarding data communication in the microservice architecture. They have mentioned data streaming services between the databases with the help of message brokers like Kafka [4].

There are two service-to-service interaction styles: synchronous communication and asynchronous communication. Most systems use synchronous communication for inter-service communication [5]. One service makes a request and waits until the next server responds to that request. This is defined as synchronous communication. gRPC and Representational state transfer (REST) application programming interfaces (API) are the most common mechanisms used in microservice architecture to conduct synchronous communication. The main advantage of this method is that developers can build the solution with straight forward using this method [6].

gRPC is an open-source communication developed by Google, which possesses a high-performance protocol and is light-weighted and modern. gRPC uses the protocol buffer and uses the same TCP connection for communication [7]. In the distributed system, inter-process communication is conducted using the remote call procedure mechanism. This protocol uses the same concept to exchange data between two services. When the service sends the request to the other service, that particular service gets held up until a response is received from the other service. Hence, all gRPC clients tend to work in synchronous communication. REST is a very popular communication protocol used among software developers as it is straightforward to implement, and many resources are available in each programming language to use the REST protocol. REST protocol is mainly used for creating APIs. By using REST APIs, people can achieve better scalability, simplicity, portability, and modifiability [8]. The main part of the REST is that request, and the response belongs to the client and the server.

Asynchronous communication is conducted using the concept of messaging. In the early days of asynchronous communication, people used message brokers. However, it is unnecessary presently. Unlike REST, WebSocket supports asynchronous communication using a single TCP connection [9]. Currently, API also can design using the WebSocket protocol.

KTH Royal Institute of Technology researched the impact of inter-service communication in microservice [10]. They used gRPC, REST APIs and RabbitMQ method to do the inter-service communication and evaluated the throughput and the availability. They have considered the gRPC and REST APIs as synchronous communication, and RabbitMQ is considered as asynchronous communication. They have proved that asynchronous communication performs better with high availability and efficiency. K.S Gurudat and the team conducted a similar kind of research related to microservice communications [11]. In that research, they also evaluated the REST APIs and the messaging protocol using the RabbitMQ broker. According to their experimental results, a low user count of load performs best for the REST protocol. Nevertheless, when user count gets high and throughput gets high, Messaging protocols perform better in response time. However, these two research studies have not been conducted on cloud servers and are solely used on-prem computers to conduct the experiments.

Joel Fernandes and team conducted research on the performance evaluation of two well-known protocols of AMQP and RESTfull web service [12]. According to their experiment, AMQP protocol performs well when transmitting massive data between the client and server. RESTfull web service is considered more lighted and highly flexible to call the large web services [13].

Daniel Cordeiro et al. showed the energy spent in the mobile applications when executing and sorting algorithms with different types and sizes using the SOAP, REST, gRPC and Socket locally [14]. Their experimental results showed that type, size and algorithm complexity directly impacted energy consumption. The communication REST protocol is the most economical option compared to other protocols. Researchers conclude that computational offloading in mobile devices save much energy.

Alam Rahmatulloh and his team analyze the WebSocket protocol's data transmission [15]. They have used the personal computer as a server and smartphones as the client. Both devices are connected to the same local network. Data is an exchange between the client and the server using the WebSocket, and it measures the number of frames transmitted and the data usage when sending several image types such as RAW, BMP, JPEG, PNG, and WEBP with different sizes. The experimental results showed that WebSocket communication could be used without the web browser and separate web servers. However, the WebSocket JPEG file formalizes the highest-performance file extension compared to other image file formats. Their research is limited to a single user and not tested on multi-user and cloud environments. Some researcher mention that Web Sockets is the revolutionary protocol for the real-time communication [16].

Several studies have been conducted to determine the different protocols' performance from various perspectives. Recently enterprise systems are moving to event base communication [17]. Most of the researchers have used on-prem servers and local networks. But the current trend is that all the enterprise software is moving to cloud-based platforms.

This research focuses on such gaps and evaluates the most trending inter-service communication mechanisms.

III. METHODOLOGY

This research elaborates on identifying the most trending communication protocols and evaluates those protocols by considering the overall application performance in terms of response time in the cloud-based environment. We have chosen the two synchronized protocols, REST and the gRPC, and one asynchronous framework, the WebSocket, with the literature review results. There is no sustainable method for choosing the inter-service communication mechanism in the microservice environment. Hence, the developers communicate with the existing method or the REST protocol. This research is an arena to gain more ideas about inter-service communication methods and their impact on application performance. When converting the monolithic system into microservices, people can't change the interface which they have exposed to external parties or other systems. Most systems use the REST protocol to expose the service to other systems. Therefore, this implementation followed the same behaviour.

Most enterprise software is developed using Java language [18]. Unlike other languages, Java supports any platform without issues through the support of the JVM. Most software development companies use the spring boot development framework to develop their Software, as Spring boot supports annotation-based programming, they can increase the overall efficiency of the developments, and it helps to avoid boilerplate codes. Also, it contains embedded servers on it [19]. Java and Spring boot have more resources for development. Hence, after scrutinizing the above facts, we have implemented the two microservices using the Java programming language and the Spring boot framework to evaluate the chosen protocols.

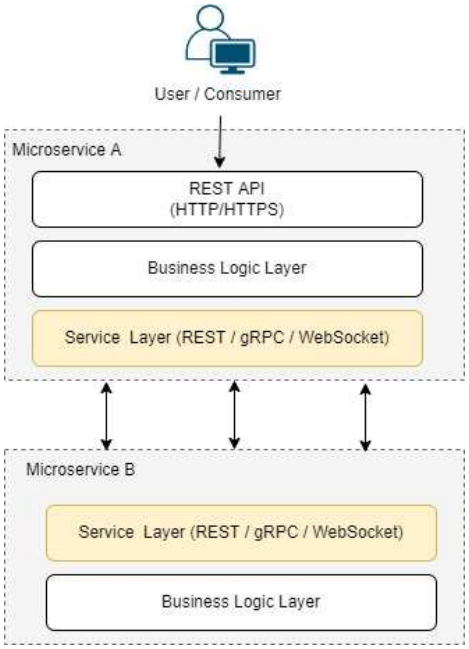

Fig. 1. Internal Software Archieture

As per figure 1 above, REST APIs are exposed from one microservice to send the request to another microservice. There is no heavy business logic built inside the microservice. In microservice A, the business logic layer is responsible for validating the request /response payload, calculating the response time and logging. In microservice B, the business logic layer generates the correct response according to the request call from microservice A.

We have logged the response time data to the log file using the business logic layer using the Log4j2 logging framework [20]. This response time refers to the exact turnaround time between the request and the inter-service communication response. This calculation does not include any other logic execution time. Figure 2 shows the internal architecture of the developed component.

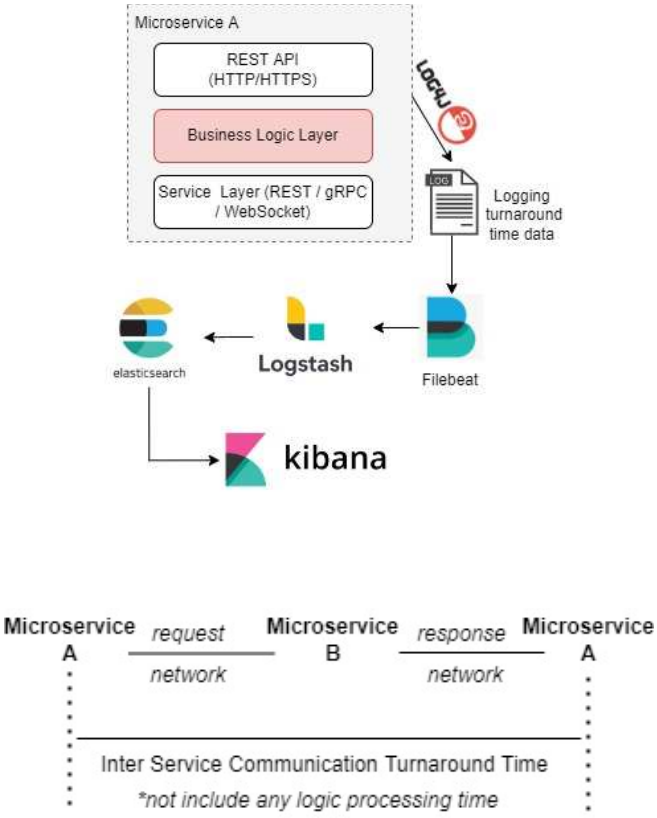

Fig. 2. How component integrates in microservice

The log file generated from microservice A is read by the file beat agent, which is separated and installed to the deployed VM in real-time. Then, the setup pipeline logs are sent to Logstash for processing with the grok patterns. We have written the grok pattern in such a manner so that it could separate the log time, response time and other details. Mandatory details are stored in Elasticsearch and displayed through the Kibana dashboard [21].

IV. RESULTS AND EVALUATION

By critically reviewing the microservice architecture and the software architectural patterns, we found that application performance quality attribute plays a significant role in the architectural decision-making process. In the microservice architecture, services are distributed, and they have to communicate with each other to produce a response to the client. Inter-service communication among the microservice has contributed to the application's overall performance in

terms of response time. HTTP communication, gRPC communication, and web socket implementation have been developed to test and evaluate the response time and the efficiency of inter-service communication.

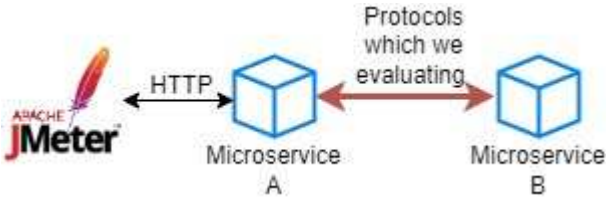

Fig. 3. High-level load test architecture

We have deployed the developed system in the well-known cloud provider AWS (Amazon Web Services) [22], as per figure 4. We have chosen AWS because most of the enterprise software is deployed on the AWS, and most of the research is conducted on the AWS standard EC2 virtual machines. Also, those machines have Intel Xeon processors that are burstable with high frequency, suitable for any general-purpose application deployments, and can balance the overall server CPU/ Memory/Network resources.

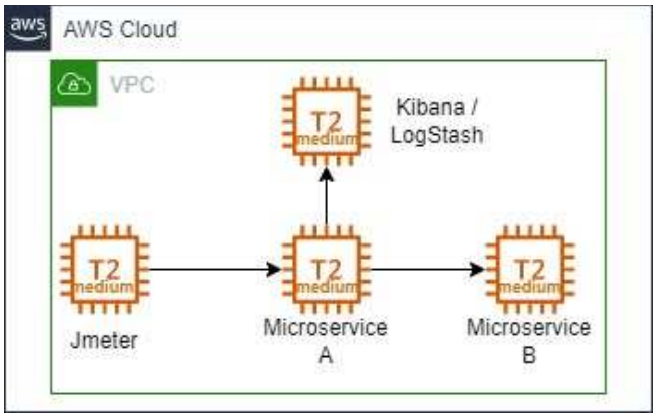

Fig. 4. Cloud deployment diagram

t2.Medium EC2 instance has two virtual CPUs and 4GB RAM to deploy the two microservices, JMeter and Log analysis Software. All the VM instances are deployed in the same VPC (Virtual Private Cloud) and the same subnet to reduce the network call latency between subnets. Apache JMeter is the tool that the industry uses to perform load tests. Hence, this research also uses JMeter to generate traffic [23]. We have executed the tests in two different ways.

1. Controlled the TPS/Payload size and measure the service-to-service communication time
2. Only Controlled the payload size and measure how service-to-service communication time affects the overall application performance in terms of throughput and response time

A. Controlled TPS/Payload Size

We have chosen different test scopes for evaluation purposes based on what other researchers have done in several studies. We have started with different size of payload sizes and different controlled TPS, and tested the below payload sizes by sending the traffic using the JMeter tool.

- HTTP GET method and as response backend microservice return empty with the 200 HTTP response code.
- HTTP POST method with 1KB size payload, and as a response backend microservice return 1KB size JSON payload with 200 HTTP response code.
- HTTP POST method with 5KB size payload, and as a response backend microservice return 5KB size JSON payload with 200 HTTP response code.

Each of the above payload sizes is tested for controlled throughputs such as 10TPS and 100TPS.

TABLE I. TESTING SCENARIOS

HTTP			gRPC			Websockets		
URL Only	1 KB	5 KB	URL Only	1 KB	5 KB	URL Only	1 KB	5 KB
10 TPS	10 TPS	10 TPS	10 TPS	10 TPS	10 TPS	10 TPS	10 TPS	10 TPS
100 TPS	100 TPS	100 TPS	100 TPS	100 TPS	100 TPS	100 TPS	100 TPS	100 TPS

The above table mentions all the test scenarios, and we run the test for 1hr for each. With this test, we collect the time spent on the service-to-service communication through the ELK stack, and from that, we get the average of the overall service-to-service communication turnaround time in milliseconds.

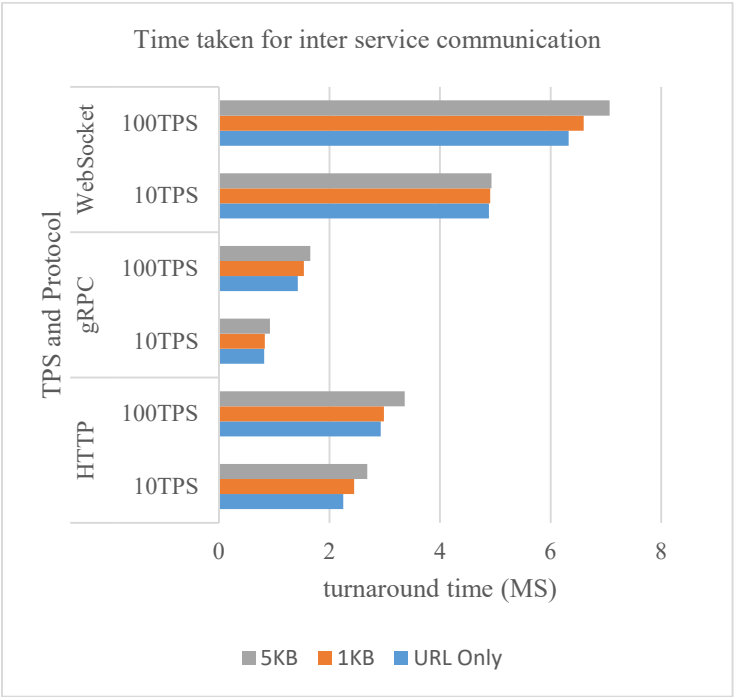

Fig. 5. Inter-service communication latency graph

Figure 5 above shows that every protocol takes a higher time when the payload size is large. This is due to the large network packets transferred from the network layer, and then the network bandwidth is allocated for those packet transfers.

By carefully considering each communication protocol's turnaround time, we can conclude that gRPC communication protocol achieves low latency when communicating the distributed services. According to the graph, the WebSocket protocol took a considerably high latency when communicating with the microservice to microservice as request/response bases. That protocol is not matched for the inter-service communication to exchange data. Many articles show that web browser-level communication is faster when using the Web Socket protocol, but it does not fit the inter-service commutation in the distributed environment. gRPC is a remote call procedure base protocol that uses the tight packing of protocol buffers when transferring data between servers using a single TCP connection between the sender and receiver. In the HTTP protocol, we must create and close the connection between the client and the server in each request. On the contrary, it does not compress the packets when sending and receiving. Hence, HTTP protocol consumes the network layer resources more than the gRPC protocol.

B. Controlled payload size

To evaluate how inter-service communication protocol latency affects the overall application performance in terms of the overall response time and the application thought, we conducted the testing in the same cloud environment. To conduct this test, we used the average JSON payload size, which is 1KB size, and calculated the application throughput and overall application latency using the JMeter script by adding the listeners. We have calculated the average turnaround time for inter-service communication for each protocol through the Mean time from the integrated ELK stack.

According to the graph, when inter-service communication takes less time, overall response time improves. Inter-service commutation turnaround time is proportionally equivalent to the overall application response time. We can see the higher throughput recorded for the gRPC protocol. With that observation, we can conclude that the efficiency of inter-service communication in the distributed environment directly impacts the overall application throughput.

V. Conclusion

This research evaluated the most trending communication protocols and then checked the overall application performance and the time taken for inter-service communication. Most software companies migrate their monolithic architecture software to microservice-based Software to achieve performance. We know additional latency is added when communicating with the microservice in a distrusted environment; performance will be impacted. We have used industry-standard methods to evaluate the systems, and the most popular Java language framework is used to develop the microservices. The collected statistics show that gRPC protocol performs well by taking less time for inter-service communication. Hence, the application's overall performance also increased in terms of throughput and response time. We noticed that WebSocket communication is not fit for microservice's inter-service communications. Most sources show that WebSocket is suitable for communication between the UI and the services.

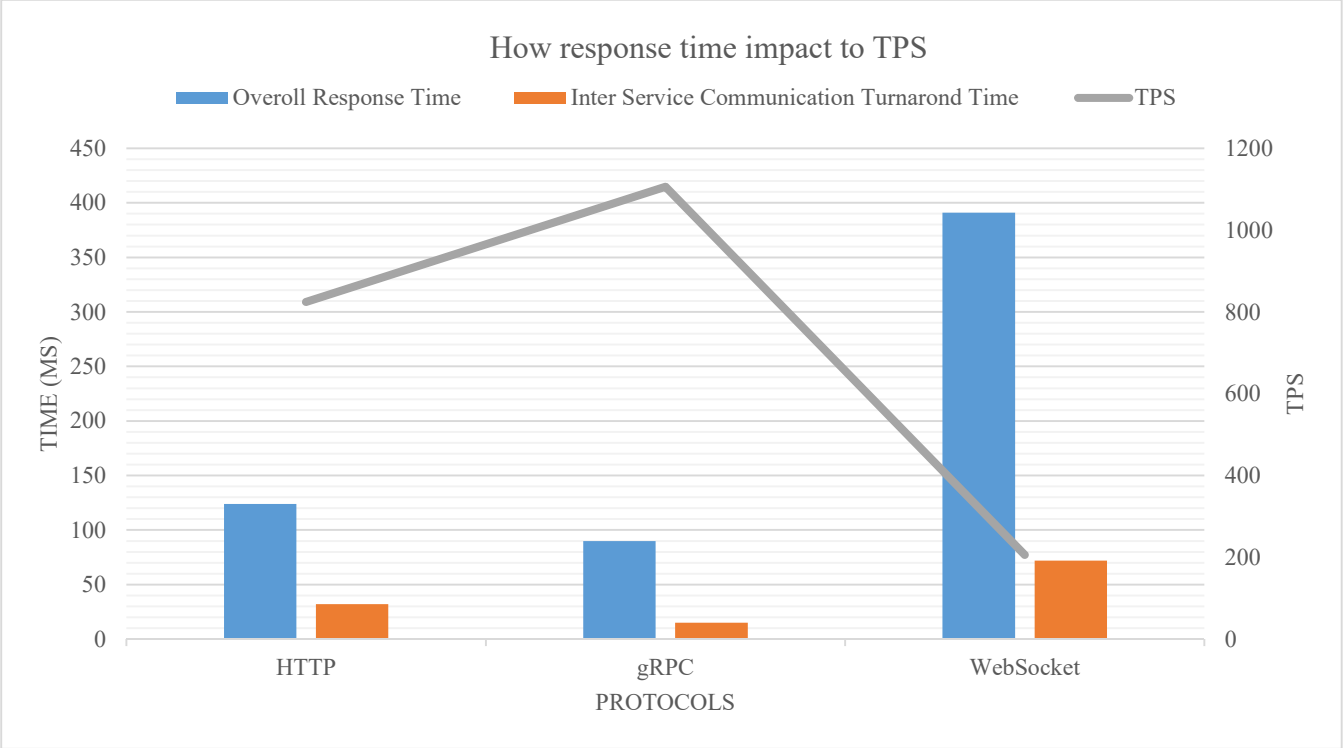

Fig. 6. Appliaction performance graph

The graph(figure 6) contains the 3 axes as used protocols on X axis, the time represented in milliseconds on Y axis, and the transactions per second count represented on Z axis.

This research paper focuses on only performance, which is often the most critical quality attribute of software architecture among scalability, testability, maintainability, security, etc. We assume processing the logs from the ELK

stack is the best option to monitor the application turnaround time when doing the performance evaluation based on the industry reference architectures.

Microservice architecture has other quality attributes which can be evaluated with this protocol comparison as a future study of this research. Further research can be work on finding a better method to improve inter-service communication in the microservice architecture.

REFERENCES

[1] S. Weerasinghe and I. Perera, "An exploratory evaluation of replacing ESB with microservices in service oriented architecture," presented at the International Research Conference on Smart Computing and Systems Engineering, Sep. 2021.

[2] V. Buono and P. Petrovic, "Enhance Inter-service Communication in Supersonic K-Native REST-based Java Microservice Architectures," p. 56.

[3] S. Li, "Understanding Quality Attributes in Microservice Architecture," in *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, Nanjing, Dec. 2017, pp. 9–10. doi: 10.1109/APSECW.2017.33.

[4] A. Smid, R. Wang, and T. Cerny, "Case study on data communication in microservice architecture," in *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, Chongqing China, Sep. 2019, pp. 261–267. doi: 10.1145/3338840.3355659.

[5] B. Shafabakhsh, R. Lagerström, and S. Hacks, "Evaluating the Impact of Inter Process Communication in Microservice Architectures," p. 9, 2020.

[6] "Building Microservices: Inter-Process Communication." https://www.nginx.com/blog/building-microservices-inter-process-communication/ (accessed Nov. 10, 2022).

[7] R. Biswas, X. Lu, and D. K. Panda, "Designing a Micro-Benchmark Suite to Evaluate gRPC for TensorFlow: Early Experiences." arXiv, Apr. 03, 2018. Accessed: Jul. 14, 2022. [Online]. Available: http://arxiv.org/abs/1804.01138

[8] A. Neumann, N. Laranjeiro, and J. Bernardino, "An Analysis of Public REST Web Service APIs," *IEEE Trans. Serv. Comput.*, vol. 14, no. 4, pp. 957–970, Jul. 2021, doi: 10.1109/TSC.2018.2847344.

[9] L. Qigang and X. Sun, "Research of Web Real-Time Communication Based on Web Socket," *Int. J. Commun. Netw. Syst. Sci.*, vol. 05, pp. 797–801, Jan. 2012, doi: 10.4236/ijcns.2012.512083.

[10] BENYAMIN SHAFABAKHSH, "Research on Interprocess Communication in Microservices Architecture," KTH ROYAL INSTITUTE OF TECHNOLOGY, 2020.

[11] G. K S and Prof. P. T, "A Better Solution Towards Microservices Communication In Web Application: A Survey," *Int. J. Innov. Res. Comput. Sci. Technol.*, vol. 7, no. 3, pp. 71–74, May 2019, doi: 10.21276/ijircst.2019.7.3.7.

[12] J. L. Fernandes, I. C. Lopes, J. J. P. C. Rodrigues, and S. Ullah, "Performance evaluation of RESTful web services and AMQP protocol," in *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*, DA NANG, Vietnam, Jul. 2013, pp. 810–815. doi: 10.1109/ICUFN.2013.6614932.

[13] C. Fu, F. Belqasmi, and R. Glitho, "RESTful web services for bridging presence service across technologies and domains: an early feasibility prototype," *IEEE Commun. Mag.*, vol. 48, no. 12, pp. 92–100, Dec. 2010, doi: 10.1109/MCOM.2010.5673078.

[14] C. L. Chamas, D. Cordeiro, and M. M. Eler, "Comparing REST, SOAP, Socket and gRPC in computation offloading of mobile applications: An energy cost analysis," in *2017 IEEE 9th Latin-American Conference on Communications (LATINCOM)*, Guatemala City, Nov. 2017, pp. 1–6. doi: 10.1109/LATINCOM.2017.8240185.

[15] A. Rahmatulloh, I. Darmawan, and R. Gunawan, "Performance Analysis of Data Transmission on WebSocket for Real-time Communication," in *2019 16th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering*, Padang, Indonesia, Jul. 2019, pp. 1–5. doi: 10.1109/QIR.2019.8898135.

[16] Bhumij Gupta and M.P. Vani, "An Overview of Web Sockets: The future of Real-Time Communication," *Int. Res. J. Eng. Technol. IRJET*, vol. 5, no. 12, Dec. 2018.

[17] K. Cebeci and Ö. Korçak, "Design of an Enterprise Level Architecture Based on Microservices," *Bilişim Teknol. Derg.*, vol. 13, no. 4, pp. 357–371, Oct. 2020, doi: 10.17671/gazibtd.558392.

[18] S. W. Perera Indika, "Taxonomical Classification and Systematic Review on Microservices," *Int. J. Eng. Trends Technol. - IJETT*, Accessed: Jul. 30, 2022. [Online]. Available: https://ijettjournal.org/archive/ijett-v70i3p225

[19] "Spring Boot." https://spring.io/projects/spring-boot

[20] "Log4j – Apache Log4j 2." https://logging.apache.org/log4j/2.x/index.html (accessed Jul. 14, 2022).

[21] P. Bavaskar, O. Kemker, A. Sinha, and D. M., "A SURVEY ON: 'LOG ANALYSIS WITH ELK STACK TOOL,'" *SSRN Electron. J.*, vol. 6, Aug. 2020.

[22] A. Bandaru, *AMAZON WEB SERVICES*. 2020.

[23] R. B. Khan, "Comparative Study of Performance Testing Tools: Apache JMeter and HP LoadRunner," p. 57.