Pakistan Journal of Scientific Research, PJOSR
ISSN (p): 0552-9050
Volume: 4, Number: 1, Pages: 1-9, Year: 2024

INTER-PROCESS COMMUNICATION AMONGST MICROSERVICES

Maria Shehzadi[1], Nauman Riaz Chaudhry[1]*, Abobakar Aslam[1], Reema Choudhary[1]

[1]Department of Computer Science, University of Gujrat, Gujrat, 50700, Pakistan
*Corresponding author: Nauman Riaz Chaudhry (Email: nauman.riaz@uog.edu.pk)

Received: 25/01/2024, Revised: 12/04/2024, Accepted: 30/04/2024

Abstract- **The purpose of the study is to perform critical analysis on Inter-Process Communication (IPC) in the Microservice Architecture and to evaluate its impact on the basis of various non-business-related functionalities, such as effectiveness of performance, accessibility, adaptability, and complexity. There are various techniques for establishing IPC within Microservices, each with its own set of benefits and drawbacks. Throughout this research, IPC approaches are divided into two categories: synchronous and asynchronous. The Representational State Transfer Application Programming Interface (REST API) and google Remote Procedure Call (gRPC) are utilized in the synchronous kind, whereas Rabbit Message Queue (RabbitMQ) is utilized in the asynchronous type. A workload test was conducted across each model to get quantitative measurements on the Performance Efficiency and Accessibility of each technique, and a relatively similar functionality set was utilized to provide qualitative data on almost every other IPC method's adaptability and complexity. The research outcome shows if there is any standardized IPC solution that can be utilized in all scenarios.**

***Index Terms*--** Microservices, Communication, Synchronous, Asynchronous, REST API, gRPC, RabbitMQ

I. INTRODUCTION

Microservice architecture is a way of breaking down applications into smaller, independent services that work separately. Each service performs a specific business function and can be created, deployed, and scaled independently. This approach offers several advantages, such as scalability, flexibility, fault isolation, ease of deployment, improved development velocity, and easier maintenance. However, adopting a microservice architecture also creates challenges, such as increased complexity in deployment and monitoring, as well as additional overhead in managing Inter-Process Communication (IPC). To successfully implement this architecture, careful design, strong development practices, and appropriate tooling are required to effectively address these challenges. This study focused specifically on IPC between Microservices. In monolithic-based systems, services may call one another at the language level, whereas in Microservices, every service continues to operate in own module, probably on a separate system from other services, and this is how IPC comes into shows an important role in the Microservice-based framework. The selection of an IPC technique is an important design choice since it will impact the efficiency and availability of the application. Numerous IPC approaches exist, each with its own implications on performance and reliability [1]. However, until today, there are no clear examples or systematic methods that can help select the proper IPC technique for the design of applications focused on microservice architecture. There is confusion regarding when to use which approach and the downside of each strategy. It is especially difficult to choose since there is no correct or incorrect choice, perhaps more or less valid based on technological and business needs. The objective for choosing this research is to associate IPC methods in the Microservices architecture from a non-functional point of view. To accomplish that target, research grounded in Microservices, using various IPC approaches, will be created. The strategy will be put through a variety of testing conditions that allow a comparative evaluation of every technique. When creating Microservice programs, the result will be actualized like a guidance for determining the correct IPC approach for the correct use of the situation.

A. Research Question

The study enquiry extracted from the description of problem, purpose/goal of intent and objective of the problem is formulated as:

RQ: How does the selection of the IPC approach influence the non-business-related requests of a Microservice dependent application?

To interrogate the study issue and then compare it against the non-business-related criterion defined in the objective section, the objective of this work is to respond to the thread that

undertake:

- In regard to system performance point of view, what would be the implication of employing different approaches for implementing IPC throughout the implementation of Microservice architecture?
- So how would the selection of the IPC approach significantly affect the service's accessibility?
- What interaction technique provides greater scalability for the system when the number of queries keeps increasing?
- Where what of the interaction technique is more complicated and takes a lot of time to be built and maintained?

B. Implications

Since there are additional elements that might impact a software systems' performance efficiency, availability, scalability, and complexity, this study solely analyses the effects of IPC on the stated quality characteristics. Furthermore, this research will not include the comparison of the error rates of IPC techniques.

II. RELATED WORK

This is intended to provide a review of current studies on the design of Microservices and, more general, the communication process of Microservices and prior research associated with data serialization strategies that can be extended to the implementation of IPCs for Microservice-based applications.

In paper [1] author said Micro services are not a perfect solution or perfect architectural approach for every software application like any other design architecture or style. Since micro services allow each part of the application to have its own technology pool, this could result toward more aggregate and complexity if somehow the various teams in an organization have to coordinate [2]. The author in the book [3] said among the most necessary and vital choices to make while setting up a system based on a Microservices architecture is how Microservices interact among each other. The researchers in [4] said that when adopting an IPC method for a micro service, it is important to take into account if the communication among them is synchronous or asynchronous. The HTTP-based REST API and gRPC are the two most used forms of synchronous communication for architects of micro services. A programming language interface expresses the series of techniques which a user can access despite maintaining the execution hidden from them. Service's API is an agreement between the service and its users in a Microservices architecture. Every service API includes a list of functions along with their names, necessary parameters, and return values [5]. RPC is a method used to allow inter process communication across many distributed systems. RPC was developed by [6] and is recognized as a protocol that allows message exchange of information between two processes while maintaining minimal overload, simplicity, and transparency. JSON operates substantially better than alternative text-based messaging formats, like XML, there are criticisms that it lacks support for namespaces and input validation [7].

In [8] the author made a detailed assessment to calculate the gap in efficiency and rate of error among both architecture. The writer built two frameworks which is alike from the point of view of market functionality but different in the architecture of software. The one system relies on monolithic architecture, and the other system relies on Microservices. In the second system, the REST API Synchronous approach was being used for IPC communication and JSON as a change of data form for every service. A test case was conducted on those systems by submitting a large volume within such a fixed timeframe. The findings of this research revealed that the architecture of Microservices may have a possibly greater fault tolerance relative to monolithic architecture, and at the same time signaling a great ability to enhance interprocess communication among services to address the duration difference by monolithic structures.

A transformation to a real-world is carried out in mission-critical research in the commercial banks by converting a monolithic related infrastructure into a Microservice related system and looking at how the Availability and Reliability improves as a part of the recent architecture. The approach consists of dividing several large components, some of which require communication with fourth providers. The study emphasized that the architecture built on Microservices has increased the availability of the desired infrastructure as the current system has been split between multiple parts and separate from one another, making it easier to load-balance separate services as required [9].

In [10] the research performed out by expertise at IBM aimed to build an architecture that is designed for executing the architecture of Microservice architecture. Researchers designed 2 parts of the model application—one based on monolithic and another relying on Microservices. Researchers found systems are more complex efficiency and higher hardware resource usage in the application's Microservice edition relative to the monolithic one. The study defines the poor architecture of process communication in the infrastructure of Microservices as the obvious change of efficiency, and hence enabled the opportunity for further study and innovation in this area. The article did not recommend a concrete approach or recommendation on how to solve these problems, and instead pointed out the possible future work for them.

Another research carried out in which the researcher contrasted the efficiency of the REST API versus the Advanced Message Queuing Protocol (AMQP), and it's among the procedure employed during message related interaction that comes underneath the Asynchronous section. The study carries out the tests by establishing two totally distinct software cases that actively receive messages for a duration of 30 minutes, with an estimate of 226 requests per second. Every case will execute the provided user request and preserve it in a permanent database [11].

The one cases was built on the REST API communication, and the second was established using AMQP. After performing

TABLE- 1 RELATED WORK

Reference	Problem	Algorithm	FutureWork/ Drawback	Results
[8]	Slow release cycles, limited scalability and low developer productivity.	RESTAPI. JSON.	Message breakers, such as RabbitMQ.	Microservice has a higher error rate but a slower response time.
[9]	Availability Reliability scalability	External API: TCP messaging queues in RabbitMQ. ForexAPI: Remote Procedure Call (RPC).	The implementation of specific techniques has been used as an argument in support of increased scalability.	Better scalability, reduced complexity.
[10]	Design an infrastructure that is improved for performing Microservice architecture.	Acme Air, for Web services, Node.js and Java Language.	Performance overhead higher hardware resource consumption Significant performance degradation. Network virtualization behind the performance gap.	No prescribed solution Performance is 79.2% low on hardware configuration. On Node.js, consumed 4.22 times more time. On Java a consumed more time in the application server.
[11]	RESTful web services versus (AMQP)-based on communication that falls under Asynchronous category.	REST API and AMQP that frequently receives messages for half an hour time period with 226 request per second.	RESTful Web service with RabbitMQ server	AMQP performs better than REST API
[13]	Most optimal message and data serialization format	XML and JSON. Protocol Buffer, and Apache Thrift.	Protocol Buffer for new systems. Use JSON for existing web services.	XML format should be avoided and use JSON for the development of existing web services.

tests, the result shows that for situations where it is ok to receive and process-intensive data, AMQP works much better than REST API, since it has an improved data loss management system, improved messaging organization, and reduced hardware resources.

In another paper the researcher have evaluated the two Synchronous Communication techniques REST API and GPRC, both in terms of the message exchange format and data serialization for Software Defined Networking (SDN) application-layer communication [12].

III. METHODOLOGY

In order to promote the achievement of this goal and to be willing to address the study question, the researcher has followed "Empirical research method" and the "Design Science Research Methodology" (DSRM) [14] for the methodology for this work. Another approach has the ability to create artefacts for solving a specific problem and simulation is itself [15].

Empirical Research methods are adopted to achieve objectives mentioned above. Empirical research can be defined as a theory which is continuously being updated by the researcher by their practices in real world or by testing their hypothesis continuously [16]. In empirical research these step follows:

- Review of literature and hypothesis formation.
- Based on hypothesis, building system using M&S techniques.
- Validating the hypothesis with the gained knowledge form literature.
- Case study formulation using M&S for testing of hypothesis.
- Evaluation of the result gained.

The Design Science Approach it encompasses the one that follows tasks:

- *Problem Recognition and inspiration:* at this point, the researcher covers the following review of related literature and explains the importance of the resolution [14].
- *Determining the objectives of the solution*: at this point, the researcher concludes aim of the solution from the description of the problem and the current understanding of

what is achievable.

- *Development and Implementation*: at this point, a demonstration concept of Microservice framework for an e-commerce set-up using various IPC approaches has been planned and implemented.
- *Presentation*: at this point, how the artifacts provided will be used to fix the issues.
- *Evaluation and Communication*: The aim is to emphasize the problem and its significance, including the use of the artefact generated to fix the issues and its uniqueness.

Briefly, two approaches are being used in this research work i.e. design science research and empirical research. This combination of two approaches used lead to a hybrid approach for the accomplishment of our objective. Figure 1 shows the steps involved in the methodology

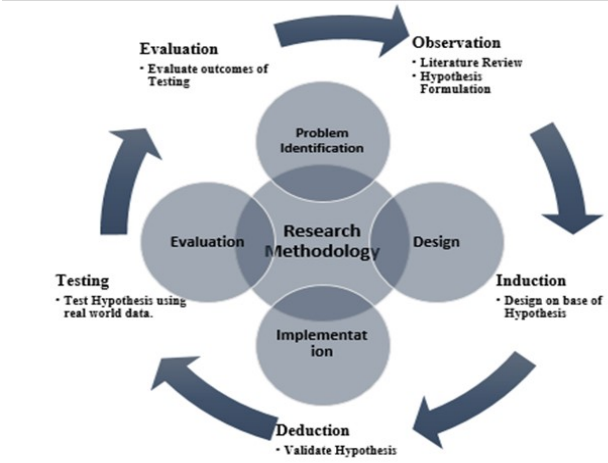

Figure 1: Steps involved in research methodology.

A. gRPC Architecture

In Figure 2. gRPC is a modernized RPC-based method designed and distributed by Google for constructing cross-language client and server applications. RPC is a method used to allow inter process communication across many distributed systems. RPC was developed by [5] and is recognized as a protocol that allows message exchange of information between two processes while maintaining minimal overload, simplicity, and transparency.

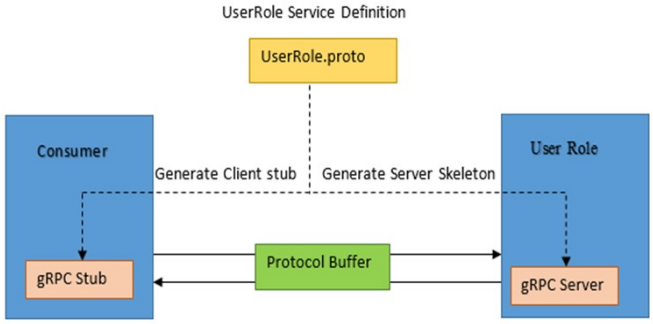

Figure 2: The Architecture of gRPC in Microservices

B. REST API Architecture

REST APIs are among the most frequent ways for two methods to share data, irrespective of their software architecture. In Figure 3 shows a system that utilises REST API for IPC communication, every service normally does have its own web-server operating on a specified port and that each service offers a collection of endpoints to allow interactions with other Microservices for data exchange.

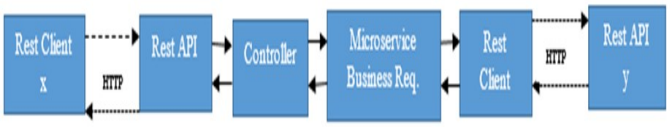

Figure 3: The Architecture of RESTA API in Microservice

C. RabbitMQ Architecture

In Figure 4. shows the client submits a message broker request. Sometime one or more services take the broker's request and execute it until the outcome is returned to the broker. Meanwhile, communication among Microservices controlled by a gateway service called a message broker which is RabbitMQ.

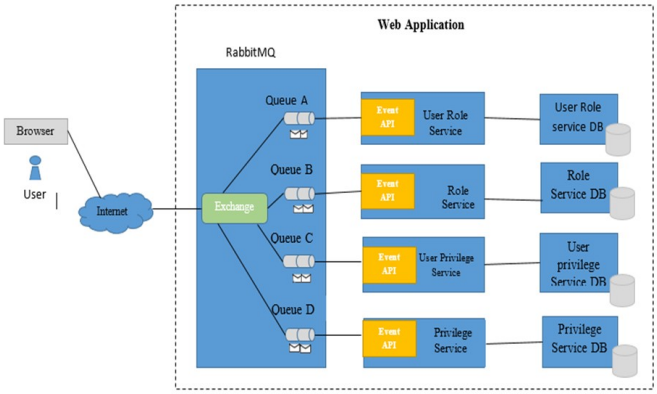

Figure 4: The Architecture of RabbitMQ in Microservices

IV. USE CASE STUDY

To represent an actual Microservices-based system, in Figure 5. A collection of features for an e-commerce scenario will be built. These Microservices will be activated in this use case:

1. Service for User role
2. Service for role
3. Service for User privilege
4. Service for Privilege.

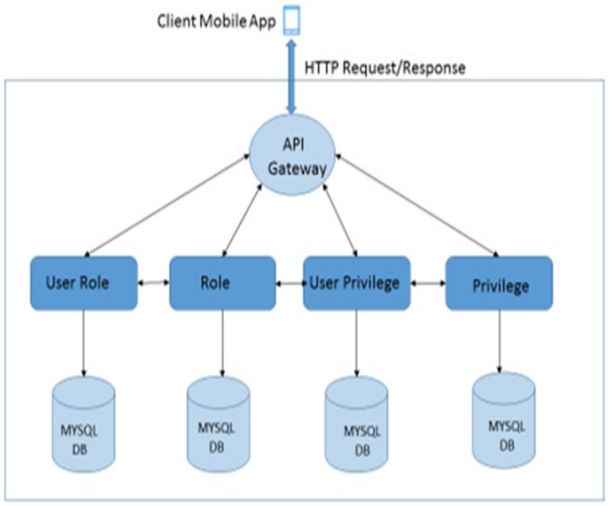

Figure 5: Architecture of Usecase

Every Microservice interacts back and forwards based on the IPC process and the data serialization format used by the system with the help of gateway. In order to connect to Microservices, an IPC method or data Serialization type used by the API Gateway retains contact with the client using Rest API and JSON formats through HTTP. HTTP and JSON are the by default protocol and data format used to interact to servers by browsers and mobile applications, whereas browsers or smart applications do not supported alternative protocols such as gRPC.

Some of this communication is generally used as a form of interaction between request and response. Through this method, one micro service sends a request to some other service and afterwards waits for that service to process the result and respond. It is typical in this form for the requester to suspend its activity while awaiting a response from the distant server.

1) REST API

This Figure 6. illustrates how IPC method works when Microservices interact with one another through the use of the REST API. The API Gateway gets user role ID and user privilege ID queries from the user's browser. This gateway then requests individual micro service via the REST API by supplying the product Id and waiting for each and every answer. Every Microservice should run a web browser under this architecture to process HTTP requests, since communication handled utilizing the REST API through HTTP protocol.

2) gRPC

The API Gateway takes the query from the client's smart

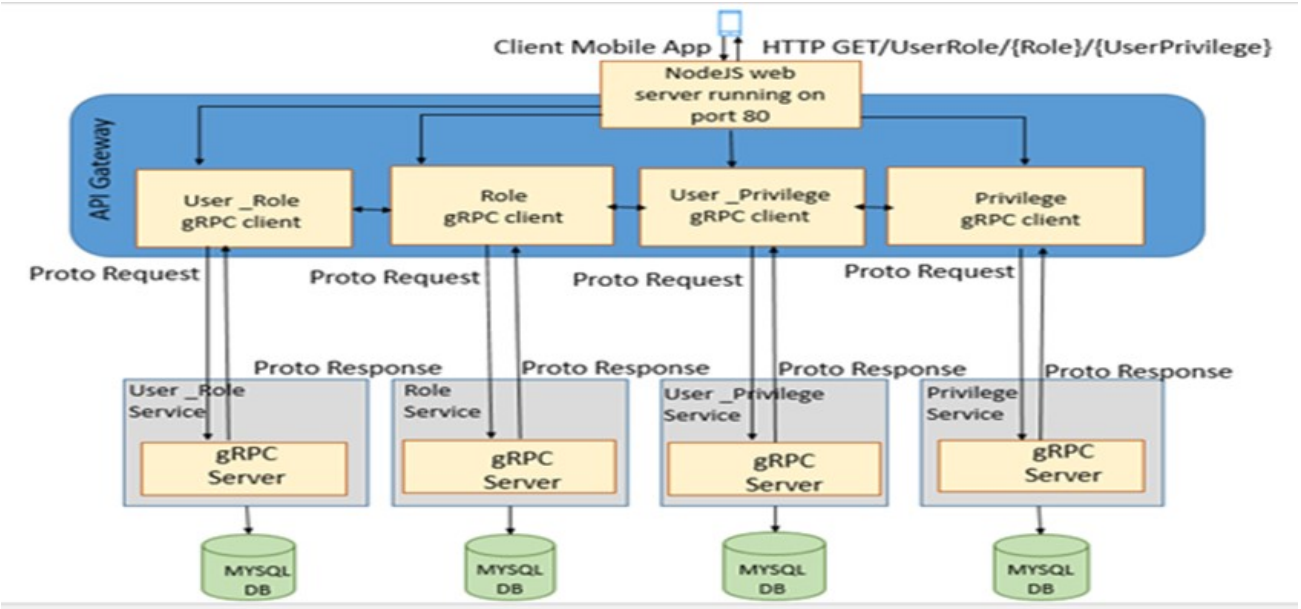

Figure 6: REST API Architecture

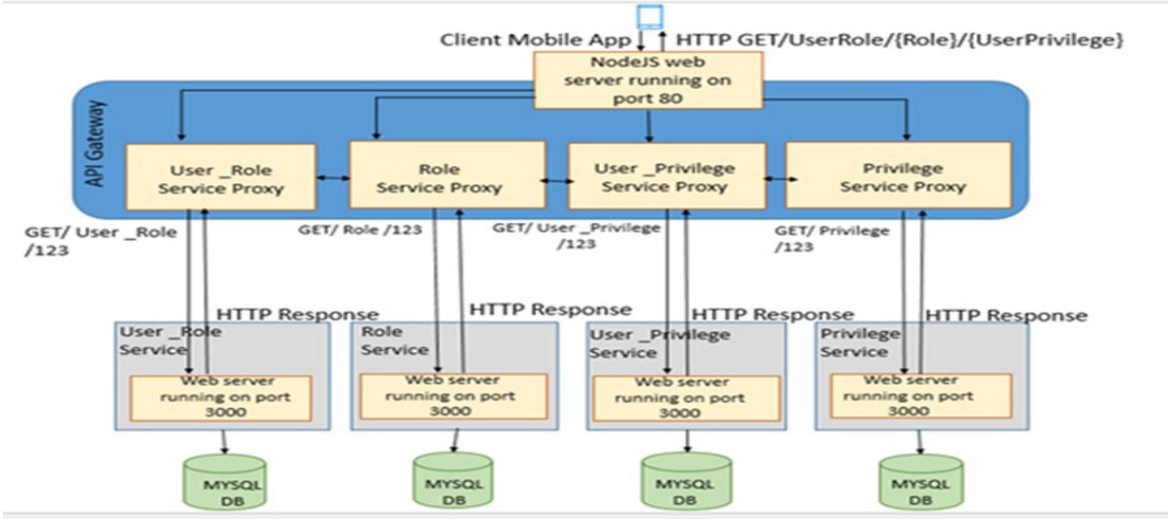

Figure 8: gRPC Architectur

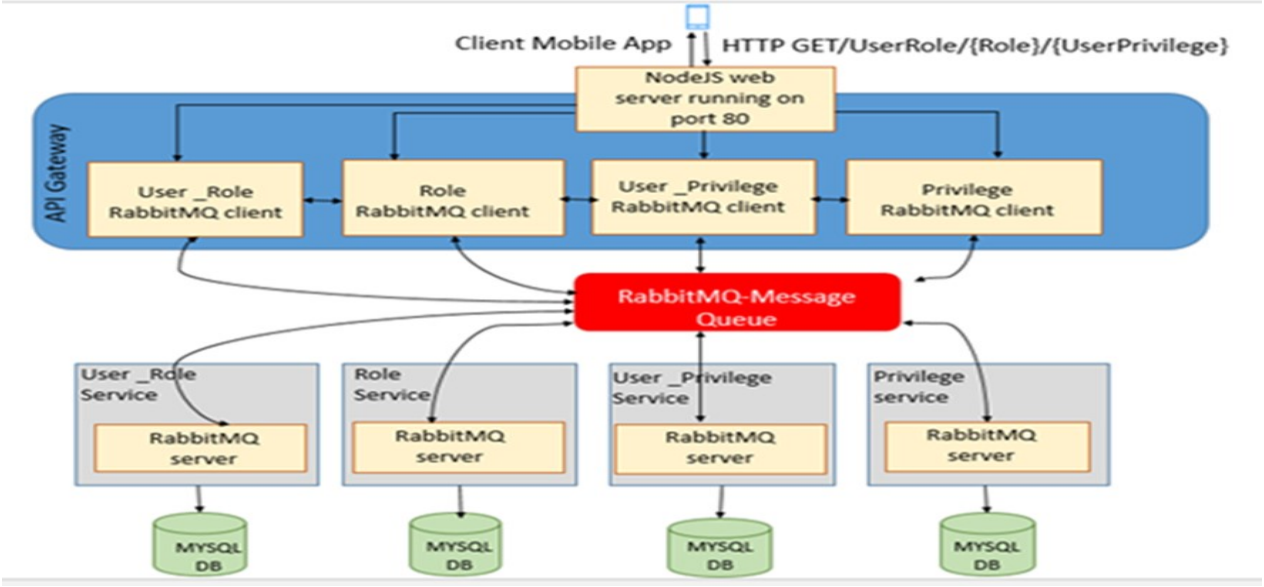

Figure 7: RabbitMQ Architectue

phone application or web browser via HTTP as shown in Figure 7, and afterwards takes that query and calls each Microservices with the necessary parameters while waiting for their answers. As gRPC uilizes a binary-based data serialization method, the outcomes delivered by Microservices via API Gateway are in bytes. As a result, the API Gateway translates the bytes and modifies it as JSON format earlier sending them to the client.

B. Setup of Asynchronous communication architecture:

In Asynchronous communication comparing with Synchronous mode one of the main distinctions is that the client will not access the server directly anymore and expects rapid response from Asynchronous communications. The client instead submits a message broker request. Sometime one or more services take the broker's request and execute it until the outcome is returned to the broker. Meanwhile, the communication among micro services controlled by a gateway

service called a message broker in the asynchronous form of communication.

1) *RabbitMQ*

Throughout the approach as shown in Figure 8, Gateway sends a query to the broker with the necessary parameters. Several Microservices would take that query, execute it, and return the data to the broker. The Gateway will accept the response since all Microservices answers are published.

V. ANALYSIS AND OUTCOME

The quantitative data serves to generalize the findings about effectiveness of performance and accessibility, whereas the qualitative data attempts to clarify the left study objective about scalability and technique complexity.

A. Quantitative Analysis

1) Effectiveness of Performance

Three test scenarios execute. The entire set of unit testing are designed to determine the latency and throughput of each IPC technique. Latency and throughput are critical factors in evaluating performance efficiency. In each scenario number of user vary. Tables 2-4 shows three test evaluations.

TABLE- 2 FIRST USECASE

IPC Technique	Time Period	Simulated Users	Mediocre reaction time	Total Call/ Answer
gRPC	60s	10	0:00:04s	1001
REST API	60s	10	0:00:07s	987
RabbitMQ	60s	10	0:00:09s	915

TABLE- 3: SECOND USECASE

IPC Technique	Time Period	Simulated Users	Mediocre reaction time	Total Call/ Answer
gRPC	60s	20	0:01:03s	1037
REST API	60s	20	0:01:07s	1007
RabbitMQ	60s	20	0:01:07s	1022

TABLE- 4: THIRD USECASE

IPC Technique	Time Period	Simulated Users	Mediocre reaction time	Total Call/ Answer
gRPC	60s	40	0:01:17s	1042
REST API	60s	40	0:01:29s	1015
RabbitMQ	60s	40	0:00:59s	1059

The time was set to 60 seconds throughout the three test evaluations. The first scenario contained 10 virtual users concurrently, the latter had 20, and the last had 40 computer-generated users concurrently. The objective of using the test timeframe as a persistent variable and the number of virtual users as a controlled parameter is to comprehend how each IPC technique behaves distinctively because the number of simultaneous requests and computer traffic to the classification grows or decreases. Throughput is determined in each test case by the total amount of requests and answers provided by the technique during the stated time limit of 60 seconds; the greater the number of requests, the higher the throughput and the better it is. Conversely, latency is assessed by the amount of time it takes to execute every request.

The outcome resulting from the initial working statistics in figure 9 and figure 10. show that gRPC exceeded REST API and RabbitMQ during the initial case, producing 14 queries further than REST API as well as 86 queries far beyond RabbitMQ; the above demonstrates that Synchronous communication could indeed grant high speed than Asynchronous communication if the system's overall load seems to be significantly small. The total number of simulated clients inside the second example is twice the amount in the first. According to the same statistics, gRPC performs best by executing a more significant number of queries than RabbitMQ and REST API. Throughout this test, gRPC outperformed REST API and RabbitMQ by 200 milliseconds in response. This time, the latencies of REST API and RabbitMQ are comparable; nonetheless, RabbitMQ processed 15 more queries than some of its Synchronous competitors. Throughout the third example, the number of simulated clients rises fourfold over the first. Throughout this trial, asynchronous communication via RabbitMQ exceeded both other two methodologies by processing a net of 1059 queries inside its specified period, whereas gRPC completed 17 fewer queries than RabbitMQ and REST API handled 44 fewer queries to RabbitMQ.

The analysis proves a large latency difference between RabbitMQ and its two Shafts approaches. Within that testing situation, the REST API's average time to response was 22% more than RabbitMQ, while gRPC was 13% faster. This information is critical for determining substantially wide the performance gap across Synchronous and Asynchronous IPC protocols could appear when the server seems to be under heavy stress.

2) Accessibility

There are many other aspects that might influence a system's availability; maybe just hardware components can influence a system's accessibility ratio. All metrics outside of IPC were omitted for this analysis.

$$\text{Accessibility} = \frac{TTF}{TTF + TTR}$$

TTF outlooks for "Time to Failure," and TTR outlooks for "Time to Recovery". TTF denotes the amount of time the scheme is projected to be operational afore failing. TTR, on the

other hand, reflects the time it takes for the system to recover from a failure. A test case was run against all three alternative IPC techniques headed for see which individual provides the highest level of accessibility.

TABLE- 5: ACCESSIBILITY DIFFERENCE BETWEEN IPC METHODS

IPC Technique	Simulated Users	TTF (second)	TTR (second)	Accessibility
RabbitMQ	150	300s	7s	0.97
gRpc	150	180s	9s	0.95
REST API	150	120s	11s	0.91

Once the applications went down, the Kubernetes cluster that was in charge of maintaining them was manually started up. Respectively gRPC and REST API were unavailable for roughly 20 seconds after that, while RabbitMQ was unavailable for an additional 10 seconds. It is feasible to conclude that an Asynchronous strategy employing RabbitMQ provides greater Accessibility than the Synchronous competitors.

B. Qualitative Analysis

1) Adaptability

Adaptability is frequently associated with how asset use rises when program wages increase. Thus, it relates to how simple it is to allow the framework to expand its capability by acquiring additional assets. According to the findings of a prior tests performed, it's indeed feasible to conclude that Asynchronous techniques provide more adaptability beyond the package than Synchronous transmission methods. The latency increases throughout Surely enhance utilizing RabbitMQ seems to be more progressive, whereas gRPC & REST API get more unpredictable delay as a response of a program's heavy demand as shown in Figure 9 and 10. When the request volume rises, it is feasible to conclude that Asynchronous method outperforms over Synchronous method. Every query inside the Asynchronous method is routed across a centralized controller called Message Broker. RabbitMQ message stack may be set up to function as a swarm with several nodes. Its structure allows the network to grow in hopes of improving performance and better satisfying future demands.

2) Complexity

The following parameters were used to measure the complexity of every IPC approach:

- Lines of Code (LOC), and Function Points (FP).
- Software Testability.

IPC Communication based on REST API does have the fewest programming lines but to pass along or receive messages again from middleman, each function should get in touch with something like the intermediary. Because of this structural distinction, there seem to be extra factors to consider while testing and troubleshooting Microservices which employ a central server, like RabbitMQ.

VI. DISCUSSION AND FUTURE WORK

The impact of the IPC technique on security protocols is among the domains that have yet to be explored because perhaps the security of the IPC technique was not a primary concern due to this study. Nonetheless, throughout the investigation, the researcher discovered some security-related concerns that are important to highlight. The security precautions provided by each IPC technique vary primarily owing to the fundamental mechanism used to transport information. Asynchronous communication via RabbitMQ may provide a powerful authorization system with data access that comes out of the bag, but getting a similar safety mechanism via REST API needs additional development.

Bringing enhanced security mechanisms to REST API-based connection might result in a decrease in communication quality, something that does not occur via gRPC communication, while gRPC increases the level.

A. Future Work

The outcome of the analysis and development of the whole research project suggests more possibilities to broaden the study topic. These hereunder are several prospective avenues for expanding on this study topic:

- IPC methodology dependability: This theory did not evaluate the error margin among different IPC techniques, so it could help calculate the gap among IPC methods in terms of durability. On the other hand, it might be an essential component in several cases, making it worth investigating.
- Creating a more comprehensive heckle: Four Microservices were engaged throughout this situation and contacted the API Gateway. Creating an even denser network with many more Microservices that interact amongst one another without employing the API Gateway might give fresh insights.
- Maintainability of the IPC method: Overall diversity was determined by examining code lines, Feature Points, and their accessibility for validation. While calculating the diversity of the IPC mechanism, it is essential to consider maintainability.
- Assessing various IPC in a testing environment: being capable of tracking IPC performance inside an actual operating setup using actual traffic might give new perspectives for each IPC technique.
- The impact of coding and Infrastructure on IPC performance: Considering an essentially similar group of Microservices constructed using various scripting languages like Java or Python might give a considerable understanding of the effect of coding and Infrastructure upon Microservices efficiency.

VII. Conclusion

This study demonstrates that, currently, IPC represents one of the biggest issues with Microservices design, which might cause system damage due to various non-functional needs. Depending on this issue, the accompanying study topic was established:

How does the choice of IPC method impact the non-functional requirements of a Microservices-based system?

Substantial experimentation was performed upon every technique to demonstrate how well the selection of IPC might impact the system's non-functional needs. The review provides sound reasons to indicate that Asynchronous communication does have a benefit versus Synchronous communication since it provides improved performance efficiency, availability, and scalability while increasing complexity of the program and requiring extra technical work. More instructions to assist readers in choosing between Asynchronous and Synchronous type in various circumstances are presented. Finally, because the IPC technique is so important inside a Microservices design, it must be fully considered. There are many circumstances when one method of interaction is preferable than another. As a result, in quite an optimization criterion, both synchronous and asynchronous types must be used to meet the functional and non-functional needs of the different aspects.

ACKNOWLEDGMENT

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sector. The authors declare that they have no known competing financial interest or personal relationships that could have appeared to influence the work reported in this paper.

Maria Shehzadi: Problem investigation, Conceptualization, methodology, Writing – original draft, Writing – review & editing. Nauman Riaz: Supervision, Methodology, Writing – review & editing. Muhammad Abubakar Aslam: Supervision, Writing – review & editing. Reema Choudhary: writing – review & editing.

REFERENCES

[1] C. Richardson, Microservices patterns: with examples in Java. Simon and Schuster, 2018.

[2] T. Salah, M. Jamal Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," in 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST), Dec. 2016, pp. 318–325. doi: 10.1109/ICITST.2016.7856721.

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

[4] K. Bakshi, "Microservices-based software architecture and approaches," in 2017 IEEE Aerospace Conference, Mar. 2017, pp. 1–8. doi: 10.1109/AERO.2017.7943959.

[5] S. G. Du, J. W. Lee, and K. Kim, "Proposal of GRPC as a New Northbound API for Application Layer Communication Efficiency in SDN," in Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication, in IMCOM '18. New York, NY, USA: Association for Computing Machinery, Jan. 2018, pp. 1–6. doi: 10.1145/3164541.3164563.

[6] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," ACM Trans. Comput. Syst., vol. 2, no. 1, 1984.

[7] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study," Caine, vol. 9, pp. 157–162.

[8] G. Johansson, "Investigating differences in response time and error rate between a monolithic and a microservice based architecture," Master's Thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2019.

[9] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, and S. Dustdar, "Microservices: Migration of a Mission Critical System," IEEE Trans. Serv. Comput., vol. 14, no. 5, pp. 1464–1477, Sep. 2021, doi: 10.1109/TSC.2018.2889087.

[10] T. Ueda, T. Nakaike, and M. Ohara, "Workload characterization for microservices," in 2016 IEEE International Symposium on Workload Characterization (IISWC), Sep. 2016, pp. 1–10. doi: 10.1109/IISWC.2016.7581269.

[11] 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), Jul. 2013, pp. 810–815. doi: 10.1109/ICUFN.2013.6614932.

[12] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," Proc. IEEE, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[13] A. Sumaray and S. K. Makki, "A comparison of data serialization formats for optimal efficiency on a mobile platform," in Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, in ICUIMC '12. New York, NY, USA: Association for Computing Machinery, Feb. 2012, pp. 1–6. doi: 10.1145/2184751.2184810.

[14] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," J. Manag. Inf. Syst., vol. 24, no. 3, pp. 45–77, Dec. 2007, doi: 10.2753/MIS0742-1222240302.

[15] M. Bilandzic and J. Venable, "Towards participatory action design research: adapting action research and design science research methods for urban informatics," J. Community Inform., vol. 7, no. 3, 2011.

[16] J. P. Davis, K. M. Eisenhardt, and C. B. Bingham, "Developing Theory Through Simulation Methods," Acad. Manage. Rev., vol. 32, no. 2, pp. 480–499, Apr. 2007, doi: 10.5465/amr.2007.24351453