Design and Testing on Migration of Remiss-Supply in Banking System to Microservice Architecture

Alwi Maulana
Departement of Informatics
Institut Teknologi Telkom Purwokerto
Purwokerto, Indonesia
18102185@ittelkom-pwt.ac.id

Pradana Ananda Raharja
Departement of Informatics
Institut Teknologi Telkom Purwokerto
Purwokerto, Indonesia
pradana@ittelkom-pwt.ac.id

***Abstract*—The architectural migration of the banking service system from a monolithic architecture to a microservices architecture is now comprehensive. However, service applications that adapt to a monolithic architecture have many shortcomings at the time of development. This paper analyses, migration, and testing microservices architecture to meet the needs of banking services at PT. Bank Negara Indonesia with the scrum method. The Scrum method focuses on migration analysis, data inquiry, details inquiry, remis-supply, deployment, and testing. The test results on migrating banking services to microservices can be applied and have non-constant performance.**

Keywords—banking, microservices, migration, remiss-supply, testing

I. INTRODUCTION

The acceleration of digital technology in the era of the industrial revolution 4.0 can positively impact the acceleration and innovation of digital services for employees, employees, and partners. The hope is that the efforts made on this acceleration are to build flexible, cloud-based applications. So that service providers can develop services quickly and easily [1][2]. This acceleration coincides with the emergence of development paradigms such as microservices, DevOps, and cloud computing. This technology is a technology adaptation of large companies such as Netflix, Amazon, and Uber to create applications based on microservices that are robust, more adaptive, and can run on cloud and container-based platforms [3][4][5]. Based on a report from JRebel in 2022, the trend of Enterprise Java system architecture which has the most significant percentage of digital service development is microservices architecture at 32%. The second place is monolithic at 22%. The third is modular monolithic. The fourth is SOA, and fifth is the Desktop App, sixth order Serverless, as in Fig. 1 [6].

The banking system has adapted the microservices architecture and banking itself has a vital role in controlling the progress of a country's economy. A nation's economic progress depends on its banking system's success and progress. So, the banking system is the blood of the country's economy. Significant banking activity is services in cash withdrawal and deposit transactions. PT. Bank Negara Indonesia is digitally transforming its services by implementing a microservices architecture to accommodate all complex services and systems. Microservices is a solution to the problems faced by PT Bank Negara Indonesia. Microservices architecture is a distributed approach that implements applications separately into smaller parts according to existing services. Then they do not depend on each other and can be connected through the Application Programming Interface. The application of microservices to services can be developed individually and tested on each service without affecting other services or applications so that services will always be available even though they are in development, in other hand the application can use the current technology as it needs. In practice, microservices will parse applications in business processes as web services. So microservices positively impact developers in developing applications with various programming languages [7].

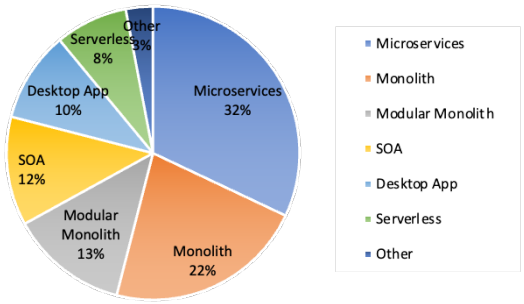

Fig. 1. Main Architectural Survey Results of Application Development.

PT. Bank Negara Indonesia has many services, including Service Inquiry Account, Cash Pick Up, Remis Supply Foreign Currency, and Remis Supply Service. Development of Remis Supply Service in the form of business processes which will later become the logical flow of domestic foreign currency storage services by each branch office. The Remis Supply business process is the result of the identification of the microservices architecture of the Remis Supply service, which previously was still implementing a monolithic architecture. The negative impact of using a monolithic architecture is that if a failure occurs in one service, it will affect other services. Each service must use the same programming language and tools or platform and cause the software to be unreliable when restarted for a development. In this case, we will migrate the remis supply service into a microservices architecture. Remis Supply Service has six subservices in the cash withdrawal deposit flow. There are three other subservices, including Inquiry Data which aims to find complete transaction data. Then, Inquiry Details to view more detailed transaction data and Remis Supply Request for

978-1-6654-6030-9/22/$31.00 ©2022 IEEE

shellfish supply to make a cash withdrawal or deposit request. The service migration process for remis supply using the scrum approach. The choice of this method is easy to handle because it is easy to handle, flexible, contains a comprehensive development strategy, and can complete complex projects with an innovative approach in a short time. The scrum master will guide this method because it applies sprints as a progress achievement target daily and weekly. Implementing the migration process using the Java programming language and PT. Bank Negara Indonesia used it to develop into microservices.

Based on the problem description, it is necessary to migrate the subservice inquiry data, inquiry details, and remis supply request service belonging to PT. Bank Negara Indonesia has become a microservices architecture using the scrum method. It then tests the results of service migration on its performance by measuring the system's reliability under various load conditions [8]. The purpose of testing the web service is with a web-service throughput of 200 threads and 1000 threads per minute and to ensure the data entered the function.

II. METHODOLOGY

A. Existing system

Remis-Supply at PT. Bank Negara Indonesia is a routine activity that aims to process inter-bank transactions. Remis-Supply is technically the stage of someone who makes a transaction to fill a balance in a savings account or take his money back. The banking system at PT Bank Negara Indonesia still uses a monolithic-based architecture that stores nominal data from bank customer activities. This service can also search data based on several parameters stored in the customer's activity history. The customer history in question includes the date of receipt, date of delivery, branch activity including withdrawals or deposits, and activity ID.

B. Scrum

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

1) *Product backlog:* This migration stage will use the scrum method. The first step in this method is to determine functional and non-functional requirements based on a priority scale by compiling the product backlog in Table 1. Five features are a high priority in migrating the Remis-Supply web service at the product backlog stage. The first features are system analysis, database design, and resource collection. Next, at the product backlog stage, subservice migration inquiry data, subservice migration inquiry details, and remis-supply requests for subservice migration. The final product backlog stage is implementation and deployment on virtual machines.

TABLE I. PRODUCT BACKLOG OF MICROSERVICES MIGRATION

No	Backlog Name	Priority
1	System analysis, database design, and collection of resources needed in web-service migration.	100
2	Tracking deposit and withdrawal activity logs (Inquiry Data).	100
3	Tracking deposit and withdrawal activity logs with details and denominations (Inquiry Details).	100
4	Request new deposit and withdrawal transactions (Remis-Supply Request).	100
5	Docker implementation and deployment.	100

2) *Sprint Planning:* This Table 2 is the sprint planning stage which aims to carry out the breakdown stage of the product backlog. Based on the backlog, there are four sprints for remis-supply services.

TABLE II. SPRINT PLANNING OF MICROSERVICES MIGRATION

Sprint	Backlog Name
System Analysis	Analysis of architecture monolithic and microservice
	Database design
	Resources collection
Inquiry Data	Creating XML Message request and response schemas
	Coding for services config web-service
	Creating repository entities and database connections
	Code implementation on CC case
	Code Implementation in the Branch case
	Testing service
Inquiry Details	Creating XML Message request and response schemas
	Making code for service config web-service
	Creating repository entities and database connections
	Code implementation in the “Inquiry” case
	Code implementation in the “Approval” case
	Code implementation in the “Reversal” case
	Testing *service*
Remis Supply Request	Creating XML Message request and response schemas
	Coding for service config web-service
	Creating repository entities and database connections
	Coding for Cut-Off Time
	Coding to process the request value
	Coding for Core Service connections
	Coding for Invoke Core Service (case “Deposit”)
	Coding for Database Log
	Coding to process response data
	Coding for remis response (Deposit)
	coding for the Supply case value set (Withdraw)
	Testing service
Docker Implementation and Deployment	Docker implementation on Inquiry Data subservice
	Docker implementation on the Inquiry Details subservice

Sprint	Backlog Name
	Docker implementation on Remis Supply Request subservice
	Deployment of all subservices

3) *Sprint Backlog:* The implementation stage of microservices migration on remis-supply services, which consists of four sprints, is as follows.

a. In sprint stage 1, which analyses the monolithic web services and focuses on system analysis, collection resources, and database design which is done with an estimated time of 104 hours starting from October 11, 2021, to October 29, 2021, as indicated on Fig 2 which is an overview of the burndown chart of sprint 1. Fig. 3 shows the results of the Remis Supply web service flow analysis, which consists of 6 subservices: inquiry data to find transaction data, inquiry details to see more complete transaction data, and remis-supply request to make new withdrawals and deposit transactions. Cash, remis-supply approval to process approval of a new trade, remis-supply reversal to process failed or rejected transactions, and remis-supply response to process responses from eligible transactions. In the microservice architecture of remis-supply based on inquiry data, inquiry details, and remis-supply requests subservices have separate businesses, as well as processes from other subservices which shown in Fig. 4. The procedure for running the remis-supply web service begins with communication using the SOAP API by sending a request in the form of an XML message by the user to the main application. Then it will be processed based on the subservice used. Then a response will appear after the process is complete.

b. The implementation phase of sprint 2 focuses on migrating the inquiry data subservice which is done with an estimated time of 255 hours starting from November 1, 2021, to December 12, 2021, as indicated on Fig. 5 which is an overview of the burndown chart of sprint 2. Inquiry details subservice aims to find transaction data with two types of cases, namely the "INQUIRY" transaction, which has an interpretation with the name case branch. Then "APPROVAL," which represents the case name cc. Fig. 6 shows the flow of the inquiry data subservice starting from the client sending a request then the service processes and performs a database search, then sends the data search result back to the client in the form of an xml response.

c. Sprint 3 is a sprint that focuses on migrating subservice inquiry details used to find detailed transaction data which is done with an estimated time of 245 hours starting from December 13, 2021, to January 24, 2022, as indicated on Fig. 7 which is an overview of burndown chart of sprint 3. There are three types of cases in inquiry details subservice, namely “INQUIRY” transactions are represented by the name of case inquiry, “APPROVAL,” which is represented by the name of case approval, and “REVERSAL,” denoted by the name case reversal. Fig. 8 shows the flow of the inquiry details subservice starting from the client sending a request then the service processes and performs a database search, then sends the data search result back to the client in the form of an xml response.

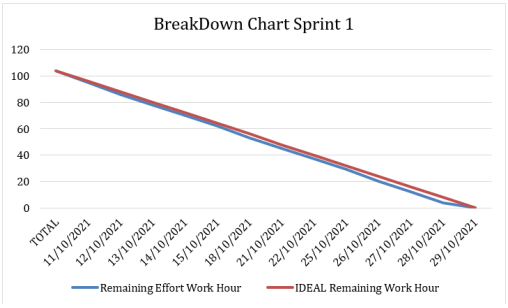

Fig. 2. Breakdown Chart Sprint 1

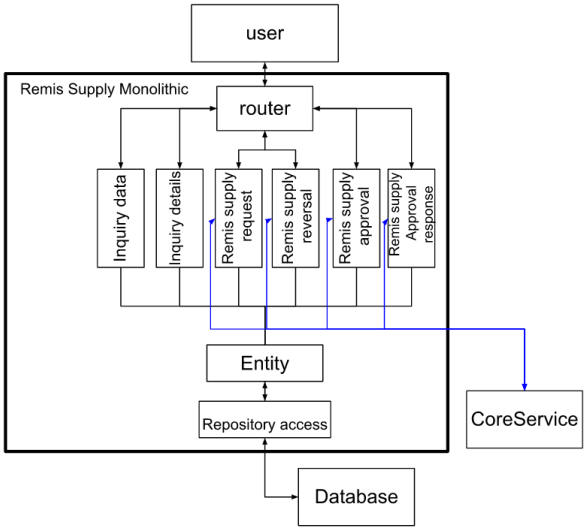

Fig. 3. Monolithic architecture-based remis-supply

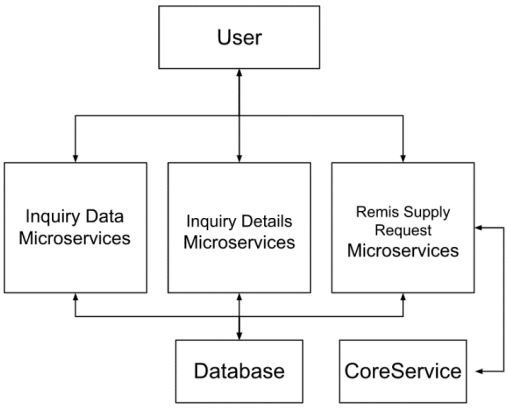

Fig. 4. Microservice architecture-based remis-supply

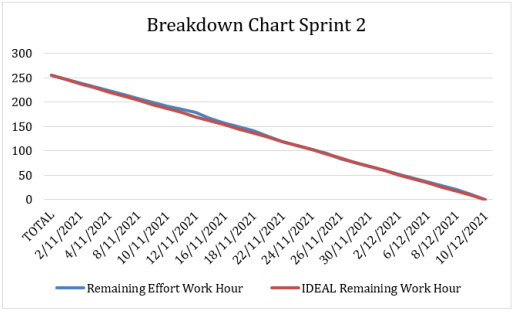

Fig. 5. Breakdown Chart Sprint 2

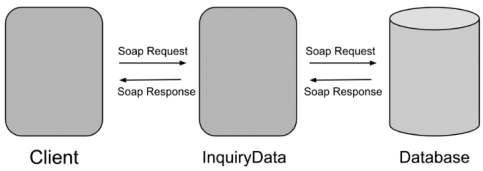

Fig. 6. Service inquiry data workflow

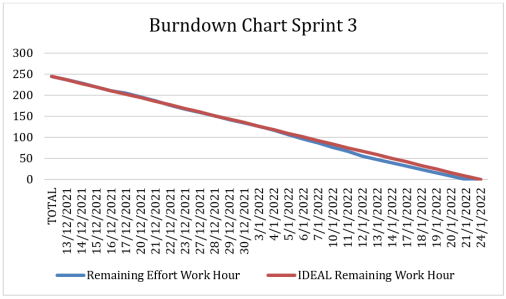

Fig. 7. Burndown Chart Sprint 3

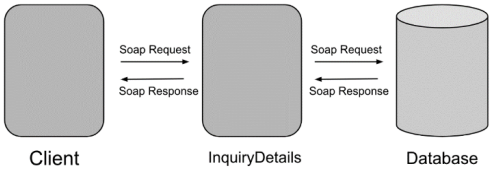

Fig. 8. Service inquiry details workflow

d. Sprint 4 focuses on migrating the remis-supply request subservice which is done with an estimated time of 306 hours starting from January 24, 2022, to March 5, 2022, as indicated on Fig. 9 which is an overview of the burndown chart of sprint 4. Remis supply request subservice has a function to make a new withdrawal or deposit transactions with two types of cases, namely "REMIS" transactions with functions to make cash deposit transactions and "SUPPLY," which functions to make cash withdrawal transactions. Fig. 10 shows the flow of the remis supply request subservice starting from the client sending a request then the service validate and performs a database search, if the data that are looking is exist in the database, then the next step is the service will send a request data to revalidated by the core service to get a journal number and save the request into database, then send the response xml to the client.

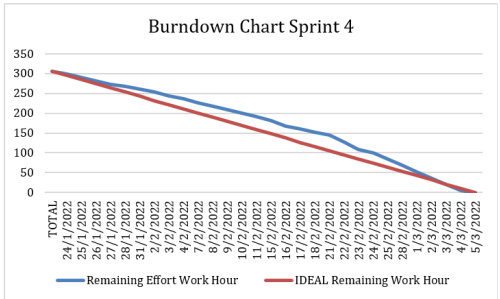

Fig. 9. Service inquiry details workflow

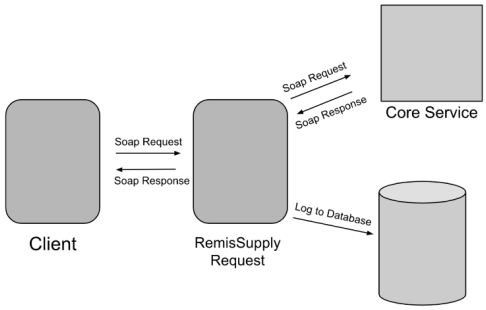

Fig. 10. Service remis-supply request workflow

e. Then after migrating all remis-supply subservices, the next step is sprint 5 which is to deploy web services on containers and virtual machine. And it is testing the performance of remis-supply web services with an estimated time of 107 hours starting from July 12, 2022, to July 29, 2022, as indicated on Fig. 11 which is an overview of the burndown chart of sprint 5.

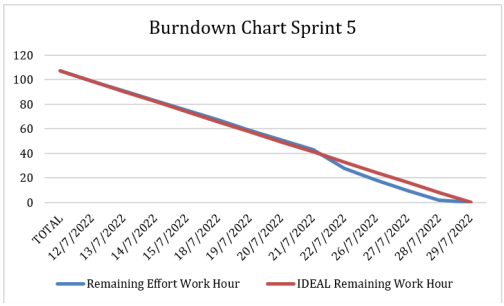

Fig. 11. Service remis-supply request workflow

III. Testing and Analysis

After running several test scenarios and collecting test results, each architecture and test shows different results based on response time, latency, throughput, and process status.

A. Load Testing Scenario between Monolithic and Microservices Architecture

1) *Monolithic:* The results of load testing on the subservice inquiry data, inquiry details and remis supply request with monolithic architecture showed different results. For example, Fig. 12 is the remis supply subservice which indicates a high response time with an average response time of 1616.06/ms, inquiry data subservice with an average response time of 389.65/ms, and the inquiry details subservice with an average response time of 241.175/ms. The test shows different result for each subservice.

Then the latency test results on the monolithic architecture show a significant difference between the subservice inquiry data and inquiry details. The one that offers the highest derivative is the remis supply request in Fig. 13 this is due to the complexity in the flow of each subservice. The same thing happened to the throughput test results shown in Fig. 14. This indicates that the inquiry details subservice has a higher throughput than the inquiry

data subservice, and remis supply request have the lowest throughput. Based on the tests carried out on the monolithic architecture, it shows that the subservice has good performance, but there are similarities in the response time results of the two, which produce graphs that are not constant.

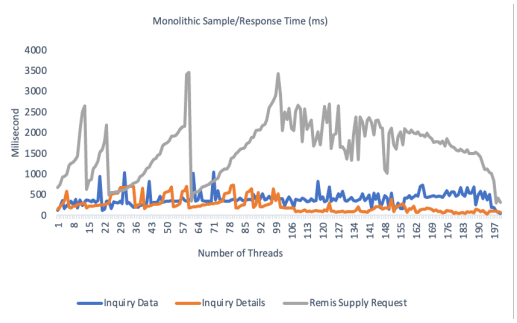

Fig. 12. Monolithic sample response time test

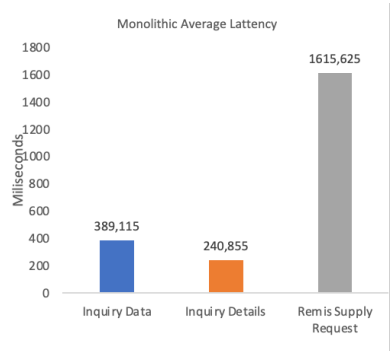

Fig. 13. Monolithic Average Latency

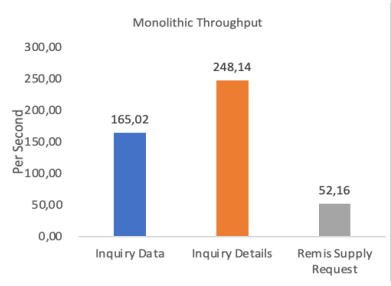

Fig. 14. Monolithic Throughput

2) *Microservices:*The load testing results on subservice inquiry data, inquiry details and remis supply request with microservices architecture showed significant differences. As shown in Fig. 15, the remis supply request subservice response time is higher than the inquiry data and inquiry details with an average response time of 42404/ms, inquiry data subservice with an average response time of 586/ms, and inquiry details subservice with an average response time of 552/ms. A failure influences this in the process that occurs in the remis supply request subservice. But both inquiry data and inquiry details subservice show non-constant results and tend to decrease in the 100th thread in the inquiry data and inquiry details subservices.

Then the latency results show that the remis supply request subservice has a higher average latency, as shown in Fig. 16 which can be seen that the inquiry data and inquiry details subservice have almost the same value. The same thing happened to the throughput results, which had differences between the subservice inquiry data, inquiry details, and remis supply request with the lowest throughput listed in Fig. 17.

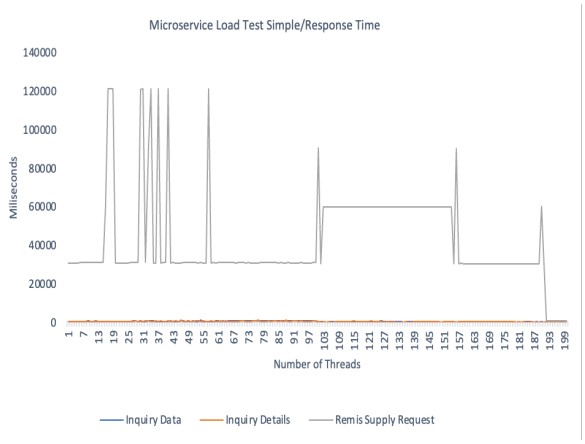

Fig. 15. Microservices Load Test Performance Throughput

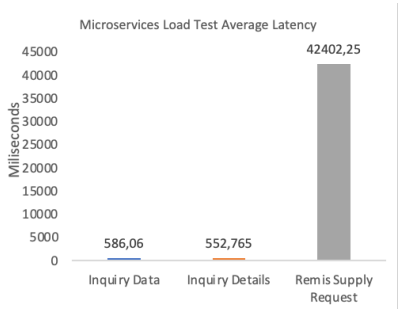

Fig. 16. Microservices load test average latency

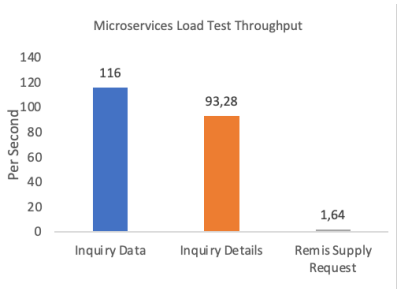

Fig. 17. Microservices load test throughput

Based on tests conducted on the microservices architecture, it shows that the remis supply request subservice has lower performance than the inquiry data and inquiry details subservice. The effect is on the remis supply request subservice, which has a more complicated flow and must check the database repeatedly and then save the data to the database. However, there are similarities in the response time results of the two, which produce unstable graphs.

B. Stress Testing Scenario

The stress test implementation aims to discover that the microservices architecture service can handle large requests. In this test, the ramp-up time was 0.1 seconds, and the loop count was 1. Additionally, setting the number of threads to 1000. Stress testing results on subservice inquiry data, inquiry

details, and remis supply request. As shown in Fig. 18, the response time for stress testing is higher than for load testing. It is also affected by some subservices that cannot accept large-scale request loads, which can fail. Remis supply request subservice has a higher result with an average response time of 1616.06/ms, inquiry data with an average response time of 389,65/ms, and inquiry details with an average response time of 241,175/ms.

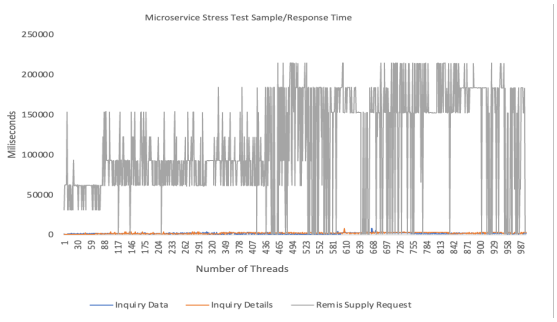

Fig. 18. Microservices stress test sample

Then the latency results show that the stress testing results have a higher average latency on the remis supply request subservice higher than inquiry data and inquiry details subservices, as shown in Fig. 19.

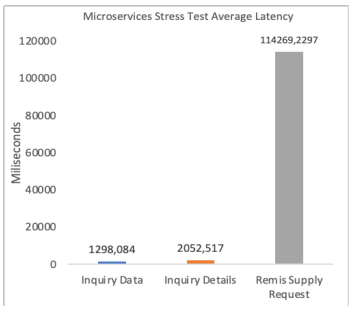

Fig. 19. Microservices stress test average latency

The same thing happened to the throughput results, where the stress testing results on the remis supply request subservice had a higher throughput than the inquiry data and inquiry details subservice shown in Fig. 20.

The tests conducted on the subservice inquiry data, inquiry details, and remis supply request with microservices architecture using stress testing scenarios show that services perform differently in large-scale handling loads. The occurrence of a failure in the inquiry data subservice and the higher is the remis supply request which show that the remis supply request is unable to handle large-scale request.

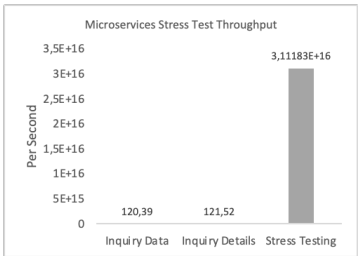

Fig. 20. Microservices stress test average latency

IV. Conclusion

After analysing the results of each test scenario. Load testing monolithic and microservices architecture, and stress testing on microservices architecture. The conclusion is that both have identical test results that are not constant. In this case study, the testing implementation uses different server specifications, so there are significant differences in test results. On a monolithic architecture with on-premises server specifications, Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz RAM dev docker 8GB server with IBM enterprise has better performance than a microservices architecture that uses a virtual machine google compute engine with specifications n1-standard- 2 (Intel Haswell) 1 vCPU 7.5GB memory and spring boot framework. But in terms of development and maintenance, the microservices architecture is preferred because the system is simpler and unrelated to each other so that each subservice can use different technologies as needed, compared to the monolithic architecture which still unites all subservices into one service.

References

[1] A. Megargel, V. Shankararaman, and D. K. Walker, "Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example," in *Computer Communications and Networks Software Engineering in the Era of Cloud Computing*, 1st ed., M. Ramachandran and Z. Mahmood, Eds. Leeds: Springer, 2020, pp. 85–108. doi: https://doi.org/10.1007/978-3-030-33624-0_4.

[2] Maniah, B. Soewito, F. L. Gaol, and E. Abdurachman, "A systematic literature Review: Risk analysis in cloud migration," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 3111–3120, Feb. 2021, doi: 10.1016/j.jksuci.2021.01.008.

[3] V. Velepucha and P. Flores, "Monoliths to microservices-Migration Problems and Challenges: A SMS," in *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*, Mar. 2021, pp. 135–142. doi: 10.1109/ICI2ST51859.2021.00027.

[4] L. Baresi and M. Garriga, "Microservices: The evolution and extinction of web services?," in *Microservices: Science and Engineering*, 1st ed., A. Bucchiarone, N. Dragoni, S. Dustar, P. Lago, M. Mazzara, V. Rivera, and A. Sadovykh, Eds. Cham: Springer, 2020, pp. 3–28. doi: 10.1007/978-3-030-31646-4_1.

[5] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li, "Microservices: Architecture, container, and challenges," in *Proceedings - Companion of the 2020 IEEE 20th International Conference on Software Quality, Reliability, and Security, QRS-C 2020*, Dec. 2020, pp. 629–635. doi: 10.1109/QRS-C51114.2020.00107.

[6] JRebel, "2022 Java Developer Productivity Report," 2022. Accessed: Jul. 14, 2022. [Online]. Available: https://www.jrebel.com/resources/java-developer-productivity-report-2022

[7] F. Rademacher, S. Sachweh, and A. Zündorf, "A modeling method for systematic architecture reconstruction of microservice-based software systems," in *25th International Conference, EMMSAD 2020*, Apr. 2020, vol. 387 LNBIP, pp. 311–326. doi: 10.1007/978-3-030-49418-6_21.

[8] F. Tapia, M. ángel Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis, "From monolithic systems to microservices: A comparative study of performance," *Applied Sciences (Switzerland)*, vol. 10, no. 17, pp. 1–35, Sep. 2020, doi: 10.3390/app10175797.