2024 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD) | 979-8-3503-6845-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICOABCD63526.2024.10704326

Event-Driven Architecture (EDA) vs API-Driven Architecture (ADA): Which Performs Better in Microservices?

Alam Rahmatulloh
Department of Informatics
Siliwangi University
Tasikmalaya, Indonesia
alam@unsil.ac.id

Fuji Nugraha
Department of Informatics
Siliwangi University
Tasikmalaya, Indonesia
fujinugraha16@gmail.com

Rohmat Gunawan
Department of Informatics
Siliwangi University
Tasikmalaya, Indonesia
rohmatgunawan@unsil.ac.id
**corresponding author*

Irfan Darmawan
Department of Information Systems
Telkom University
Bandung, Indonesia
irfandarmawan@telkomuniversity.ac.id

Erna Haerani
Department of Information Systems
Siliwangi University
Tasikmalaya, Indonesia
erna@unsil.ac.id

Randi Rizal
Department of Informatics
Siliwangi University
Tasikmalaya, Indonesia
randirizal@unsil.ac.id

***Abstract*—Monolithic application architectures are widely abandoned and replaced by microservice architectures that are more flexible and able to adapt to rapid technological changes. However, HTTP synchronous protocol-based data communication or API-Driven architecture (ADA) microservices commonly used in microservice architectures can reduce system performance when the number of services increases. To support a growing business, the scalability aspect of microservices needs to be considered. The solution in this study uses asynchronous data communication between event-based microservices or Event-Driven Architecture (EDA). Container technology is applied in this study to support the scalability aspect. Measurements of response time, error rate, and CPU usage were carried out in experiments to determine the performance of microservice architectures by implementing ADA and EDA-Driven data communication. The main problem in this study, which of the two services has the best performance? Testing and analysis were conducted in this study to reveal the comparison of their performance. The experimental findings in this study indicate that the response time of microservice architecture for EDA-Driven data communication is 30% faster than ADA-Driven data communication. There was a decrease in error rate of about 7% and a decrease in CPU usage of about 4% when EDA-Driven data communication was implemented in microservice architectures, compared to ADA-Driven data communication.**

Keywords—API-Driven Architecture, Event-Driven Architecture, Container Technology, Microservices

I. Introduction

The development and movement of technology are getting faster by entering the industrial era 4.0, causing monolithic-based application architectures to be abandoned. Monolithic architecture is no longer relevant to the current conditions, which, according to experts, have the characteristics of volatility, uncertainty, complexity, and ambiguity known as VUCA. Rapid change, unpredictability, lack of a causal chain, and a blurring of reality are characteristics of the VUCA world condition [1]–[3]. So, a new mechanism is needed to overcome monolithic architectures' capability and scalability problems.

Microservice architecture is one solution to overcome this problem. Microservice architecture is increasingly used because it offers high flexibility, can adapt to technological changes, and can help simplify developers' work [4]. Statistics from the 2021 International Data Corporation (IDC) survey, which predicts that 80% of cloud-based apps would be created with a microservices architecture, corroborate this claim [5]. The microservices architecture breaks extensive services into small services that stand alone without being dependent on one another so that development can be carried out by different teams and using different programming languages, databases, and operating systems. Many large companies have started developing their applications towards a microservices architecture, including Amazon, LinkedIn, Netflix, Spotify, SoundCloud, and others [6]. Furthermore, microservices have been the subject of numerous studies, such as those on the Internet of Things (IoT) [7]–[9], DevOps [4], and other topics.

However, behind all the advantages of this microservices architecture, there are still some areas for improvement, including a separate small service mechanism that causes communication via synchronous HTTP between these services to burden the server, thereby reducing system performance [5]. HTTP synchronous communication is usually used in microservices with (Application Programming Interface) API-Driven Architecture (ADA) [10]. Another problem in microservices that needs to be considered is their scalability and performance [11].

Several previous studies have solved microservices' problems, including container technology, which can provide easy scalability operations by multiplying service copies to overcome scalability problems [12] and elasticity [6]. Docker containers implement containerization techniques with representative technologies that have the characteristics of being lightweight and also being able to run multiple microservices so that it can improve the performance of microservices by utilizing higher resources [13].

Other researchers ran tests to examine how well RabbitMQ and RESTful API performed on microservices-based web applications. This study found that if many users send requests simultaneously, RabbitMQ is more stable than using the RESTful API [14]. When handling massive amounts of data when no response is anticipated, microservices can interact with each other using event-driven architecture (EDA) [15].

Meanwhile, this research will try to combine the two solutions that have been done by previous research by applying container technology to microservices and integration with EDA so that the proposed integration model is expected to be able to handle communication problems between services and scalability performance problems on microservices. In addition, in this study, a performance comparison will be made between event-driven and API-driven architectures on microservices.

II. The Material

A. Microservices

Microservices are the smallest collection of independent services but communicate with each other to form a single complex application unit [16]. Developers are more flexible in carrying out construction in a service because it will not interfere with other services. This is one of the advantages of a microservices architecture, in contrast to traditional or monolithic architectures, which are more complicated in development because all logic, programming, and databases are integrated into one unit. Different technology stacks can be used to construct microservices architectures, and they can implement independent technologies [17]. Based on business capabilities, each microservice operates autonomously and uses a lightweight mechanism to connect with others inside the application. The self-managing architectural style known as microservices is an evolution of service-oriented architecture (SOA) [18].

B. Container

The container is a virtual container that can accommodate various services directly and can run on the host operating system [19]. To run an application or service, containerization is a wrapper for the source code, configuration, and all connected services, including libraries and dependencies. Container technology differs from operating system virtualization, which requires an operating system in virtualization. Thus, container technology becomes lighter, portable, and independent and can run on various platforms [20]. A few benefits of utilizing this container are that it can simplify the use of microservices, swiftly scale, remove, and deploy; boost flexibility by using various frameworks and technologies, and strengthen the system's resilience [21].

One of the most popular uses of container technology is Docker, which is based on the Linux kernel. Docker containers are designed to encapsulate application source code and all its supporters into a docker image. The encapsulated application can be run on different machines with the help of the Docker container [22]. Docker images are created through configuration in a file called the docker file.

An alternate tactic is provided by Kubernetes, an open-source framework for managing workloads, services, and containerised applications. In order to make containers portable and adaptable, Kubernetes is intended for container management, deployment, scaling, and operating applications on a container [20], [23]. Aside from that, Kubernetes has a scaffold. These command-line tools can be used to manage workflow and create, push, and launch a service or application. Furthermore, a scaffold can be used on local or distant Kubernetes clusters when designing an application [24].

C. Event-Driven Architecture (EDA)

Event-driven architecture (EDA) is one of the data communication architectures that can be implemented in microservices. EDA works based on events, event publishing, and listening exchange of information or data. The EDA ecosystem broadcasts to every service that requests (event listen) or services that will receive related events [25]. Communication in EDA uses message events and works asynchronously, while API-Driven Architecture (ADA) works synchronously using API calls [10]. Network instruction detection, the stock market, sensor networks, the Internet of Things, healthcare monitoring, and mobile and wearable computing are just a few industries where EDA is frequently employed. In addition, EDA can provide advantages in developing distributed systems that are more flexible and have high concurrency [26].

The event flow in EDA is described in Fig. 1; when a request comes in, the load balancer forwards and directs it to the destination service, then the destination service will process the request and publish the event to the event bus. In addition to processing requests and synchronising data, other services also receive event notifications. The client will return the response after the request has been successfully processed. The event bus facilitates communication between components with the publish-subscribe pattern. The event bus easily exchanges data between behind-the-scenes components such as activities, services, threads, and fragments. The Event Bus offers a solution in a centralized way to notify application components related to specific events, and the most exciting thing is that there is no direct coupling between the code that acts as a publisher and the code that acts as a receiver so that changes to the code in the system are not too significant. In addition, EventBus is very flexible, keeps the system architecture decoupled, and does not require interfaces or callbacks for asynchronous communication. RabbitMQ [14], NATS Streaming, or Apache Kafka [25] can all be used to establish event buses.

III. Experimental Setup

The research method is divided into five stages: literature study, system analysis, identification of hardware and software needs, system design, and measurements, as shown in Figure 1.

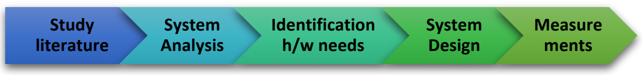


Fig. 1. Research Methodology.

A. Study Literature

The activities at this stage include reviewing existing research and studying all information related to microservices. Reference sources related to container technology, event-driven architecture, and API-driven architecture were also studied, especially relevant published journals.

B. System Analysis

To find out the performance of EDA and ADA in this study, an application prototype was created that adopted a company's business processes with many application services. The information system is depicted in Figure 3, which consists of four primary services, namely auth (service

1), cloth (service 2), stock (service 3), and sale (service 4), as well as the NATS streaming service (event-bus). All services are connected to the database used, namely MongoDB. The scenario for the auth service is to handle all authentication and user management activities. Cloth services handle the entire fabric management process. Stock services handle calculating incoming and outgoing fabrics (stock-in and stock-out). Service sales handle all sales and expenditure activities of goods. The NATS streamer, the final service, serves as the major conduit for data exchange between the four main services—publishing (send data) and listening (get data).

C. *Identification of Hardware And Software Needs*

The measurement process in this study requires hardware and software identification. The hardware specifications used are in Table 1, and the software specifications are in Table 2.

TABLE I. HARDWARE SPECIFICATION

Detail	Description
CPU	AMD A4-9120 RADEON R3, 4 COMPUTE CORES 2C + 2G @ 2x 2.25GHz
GPU	AMD STONEY
RAM	8GB
Kernel	x86_64 Linux 5.10.70-1-MANJARO
Disk	110GB
Operating System	Manjaro 21.1.6 Pahvo (Linux)

TABLE II. SOFTWARE NEEDS AND SOFTWARE ARCHITECTURE

Name	Description
Docker Lates Version	Application-oriented container
Kubernetes v1.22.2	Container orchestrator
Skaffold v2 beta20	Kubernetes support
NATS Streaming v0.22	Event bus
Typescript v4.4.4	Programming language
Node.js v17.1.0	Runtime environment
Express.js v4.17.1	App framework
MongoDB v5.0	Database
Jest v27.1.0	Testing framework
Stan.js v0.3.2	Client communication tool

D. *System Design*

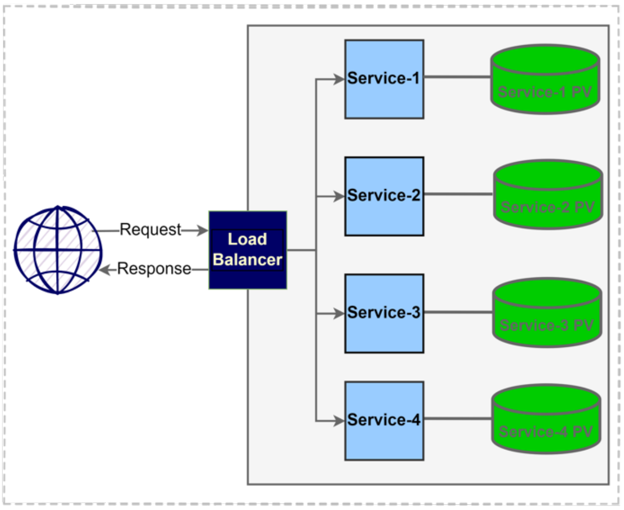


Fig. 2. ADA Communication Process in Microservice.

At this stage, a microservice-based application prototype was created by applying two different data communication principles, ADA-based and EDA-based microservices. In the ADA-based microservice architecture, an application prototype provides four services: service-1, service-2, service-3, and service-4. Each service is connected to the database, as shown in Figure 2.

In the EDA-based microservice, a prototype application is designed that provides four services: service-1, service-2, service-3, service-4, and one additional NATS Streaming service (event bus), as shown in Figure 3.

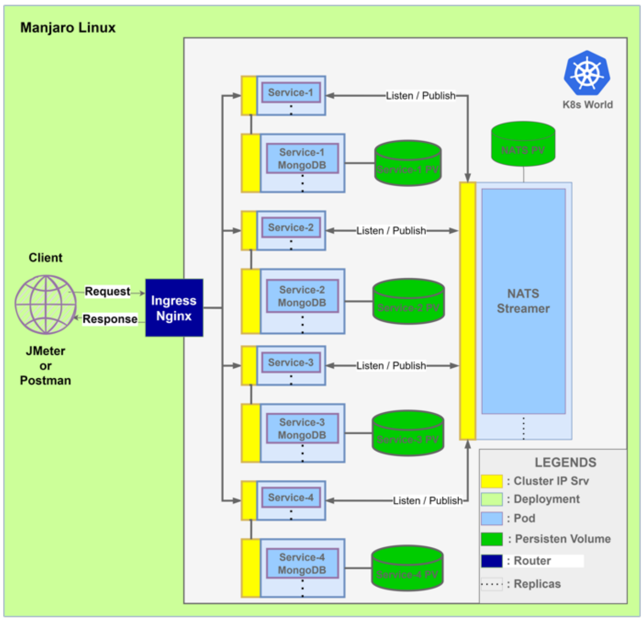


Fig. 3. EDA Communication Process in Microservice.

Figure 3 displays the data communication process architecture based on Event-Driven Architecture on a microservice-based application prototype. Each service is designed to handle different activities. In addition, the NATS Streamer acts as a liaison for the data communication process between the four services and the load balancer in publishing messages and listening to messages.

E. *Measurement*

At this stage, response times (ms), error rate (%) [14], and CPU Usage (ms) [20] are measured on ADA-based and EDA-based microservice architectures. Response time and error rate measurements are carried out with the help of Apache JMeter and the node.js API to measure CPU usage. The experiment was carried out by sending 100 request data packets from the client to an application on a microservice-based server. Each experiment was repeated ten times, and the experimental results were recorded in the table. In the final stage, the average value and the percentage change in the average value are calculated. The percentage change in value is calculated using formula (1) [27].

$$\text{Percentage} = \frac{\text{Final Value} - \text{Initial Value}}{\text{Initial Value}} \text{ x } 100\ \% \ \ldots\ldots.(1)$$

Information:

Final value	= EDA average value
Initial value	= ADA average value
Percentage	= percentage change in an average value

IV. EXPERIMENTAL RESULT

A. *Response Times Measurement Results*

The experiment was carried out by sending 100 request data packets from the client to the server, implementing ADA and EDA. It was repeated ten times, and the average value was calculated. Response times are measured using the Apache JMeter tool, as shown in Figure 4.

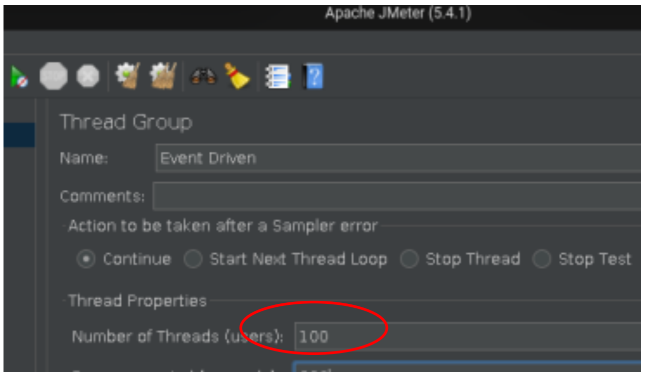

Apache JMeter (5.4.1)
Thread Group
Name: Event Driven
Comments:
Action to be taken after a Sampler error
● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test
Thread Properties
Number of Threads (users): 100

Fig. 4. Configure the number of requests on JMeter.

After ten trials, the data obtained was recorded, and then the percentage change in data between ADA-based and EDA-based data communication architectures was calculated, as shown in Table 3.

TABLE III. RESPONSE TIME RESULT

Experiment	ADA (ms)	EDA (ms)	Percentage of Change
1	616	427	31%
2	606	432	29%
3	604	422	30%
4	606	439	28%
5	619	430	31%
6	606	423	30%
7	604	422	30%
8	628	424	32%
9	623	447	28%
10	606	436	28%
Average	**611**	**430**	**30%**

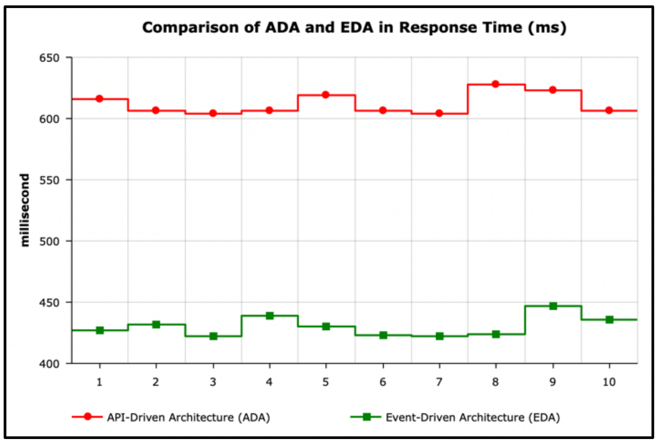


Fig. 5. Response Time Result.

Table 3 displays the results of measuring response times after ten attempts on the microservice architecture by implementing ADA-based and EDA-based data communications. In the microservice architecture, implementing ADA-based data communication results in an average response time of 611 ms, whereas implementing EDA-based data communication results in an average response time of 430 ms. This means the response times of EDA-based data communications on microservices are 30% better than ADA-based data communications.

Figure 5 displays the response time measurement data in a Line Chart. ADA-based data communication response times appear in the 600-650 ms range, while EDA-based data communication response times appear in the 400-450 ms range.

B. *Error Rate Measurement Results*

Error rate measurement is done by sending 100 request data packets from the client to the server by implementing ADA and EDA. Error rate measurement is done by using Apache JMeter tools. The percentage change in error rate data between ADA-based and EDA-based data communication architectures was calculated after the experiment was repeated ten times. Data from EDA and ADA-based error rate measurements and their percentage changes are shown in Table 4.

TABLE IV. ERROR RATE RESULT

Experiment	ADA	EDA	Percentage of Change
1	50.00%	45.00%	10%
2	50.00%	47.00%	6%
3	51.00%	48.00%	6%
4	52.00%	49.00%	6%
5	53.00%	50.00%	6%
6	53.80%	50.00%	7%
7	54.60%	51.00%	7%
8	55.00%	52.00%	5%
9	59.20%	53.00%	10%
10	59.40%	55.00%	7%
Average	**53.80%**	**50.00%**	**7%**

Implementing ADA-based data communication in the microservice architecture results in an average error rate of 53.80%, whereas implementing EDA-based data communication results in an average error rate of 50.00%. This means that the error rate of EDA-based data communication on microservices is 7% better than that of ADA-based data communication.

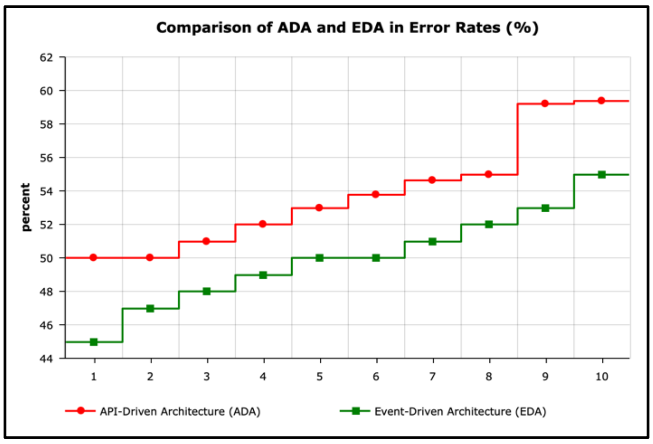
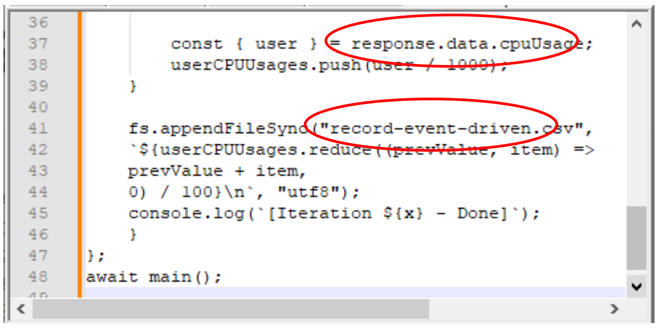


Fig. 6. Error Rate Result Line Chart.

Figure 6 visually displays the error rate measurement data in the form of a Line Chart. The error rate for ADA-based and EDA-based data communication tends to increase from experiments 1 to 10. However, in each experiment, the error rate for EDA-based data is always below the ADA-based data communication error rate.

C. CPU Usage Measurement Results

Measurement of CPU usage requires a special Javascript program code. The measurement result data is stored in a CSV file. The code snippet for the CPU Usage measurement program is shown in Figure 7.



```
36
37          const { user } = response.data.cpuUsage;
38          userCPUUsages.push(user / 1000);
39      }
40
41      fs.appendFileSync("record-event-driven.csv",
42      `${userCPUUsages.reduce((prevValue, item) =>
43      prevValue + item,
44      0) / 100}\n`, "utf8");
45      console.log(`[Iteration ${x} - Done]`);
46      }
47  };
48  await main();
```

Fig. 7. CPU Usage Measurement Program Code Snippet.

Figure 7 shows the code snippet for the CPU Usage measurement program. Line 37, **response.data.cpuUsage** is a function that calculates the CPU Usage value. Meanwhile, in line 41, there is a string **"record-event-driven.csv**," the file name for storing data on CPU Usage measurement results. The results of measuring CPU Usage based on EDA and ADA and the percentage changes are shown in Table 5.

TABLE V. CPU USAGE RESULT

Experiment	ADA (ms)	EDA (ms)	Percentage of Change
1	374.21	364.58	3%
2	397.25	362.83	9%
3	395.64	364.49	8%
4	380.21	366.55	4%
5	383.50	363.20	5%
6	380.32	375.62	1%
7	377.26	368.21	2%
8	381.07	367.44	4%
9	383.24	363.89	5%
10	389.82	383.21	2%
Average	**384.25**	**368.00**	**4%**

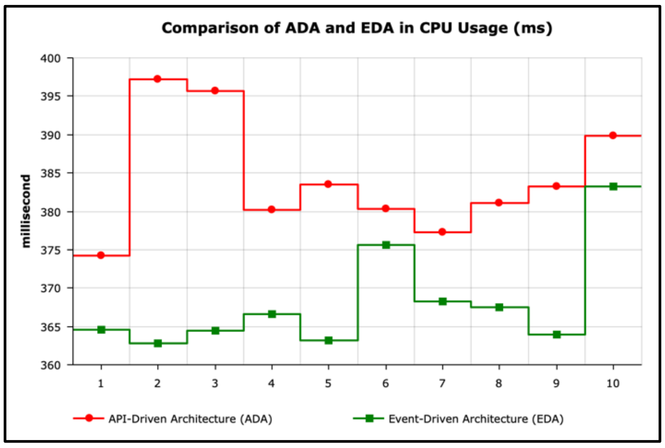


Fig. 8. CPU Usage Result Line Chart.

In the microservice architecture, implementing ADA-based data communication results in an average CPU Usage value of 384.25 ms, whereas implementing EDA-based data communication results in an average CPU Usage value of 368.00%. This means that the average CPU Usage value for data communication based on EDA on microservice architecture is 4 % better than the average CPU Usage value for data communication based on ADA. The results of the ADA and EDA CPU Usage comparison can be seen in Figure 8.

D. Comparison of the Average Value of Measurement Results

After measuring all parameters, the average value of each experiment is recorded in the table. The results of comparing the average values of each parameter are shown in Table 6.

Table 6 displays the average measurement results for each parameter from the experiments. From the percentage calculations performed, it is known that there is a decrease in the average value of each parameter. The average response time for ADA-based data communication is 611.8 ms, while the average response time for EDA-based data communication is 430.2 ms. This means that the response times of the microservice architecture, when applied to EDA-based data communications, are 30% better than the implementation of ADA-based data communications.

TABLE VI. COMPARISON OF THE AVERAGE VALUE OF MEASUREMENT RESULTS

NO	Parameter	API Driven	Event Driven	Decrease Percentage
1	Response Times (ms)	611.8	**430.2**	30%
2	Error Rate (%)	53.80%	**50.00%**	7%
3	CPU Usage (ms)	384.25	**368.00**	4%

The average ADA-based data communication error rate is 53.80%, while the average ADA-based data communication error rate is 430.2 ms. This means the error rate decreases by about 7% when EDA-based data communication is applied to a microservice architecture compared to ADA-based data communication. The average CPU Usage for ADA-based data communication is 385.25 ms, while the average CPU Usage for data communication based on EDA is 368.00 ms. This means there is a decrease in CPU Usage of around 4% when EDA-based data communication is implemented on a microservice architecture, compared to ADA-based data communication.

ADA based data communication is done with the request-response principle. The client sends a request to the server and expects a response. Each request from the client is handled sequentially by the server. This process causes waiting time and reduces system performance. While EDA based data communication can be done concurrently. The server sends an event to the client, which then reacts to the event. So that at one time several events can occur. The difference in principle in this data communication is what causes EDA's performance to be better than ADA.

V. CONCLUSION

Based on the results of the experiments that have been carried out, it is known that the response times on microservices, when applied to EDA-based data communications, are 30% better than the implementation of

ADA-based data communications. There was a decrease in error rate of about 7% and a decrease in CPU Usage of about 4% when EDA-based data communication was implemented on a microservice architecture, compared to ADA-based data communication.

The system architecture environment when testing, the number of services available to be tested, the number of requests that vary, and the tools used in testing are some challenges that can be tried in future research.

REFERENCES

[1] N. J. Pearse, “Change Management in a VUCA World,” in *Visionary Leadership in a Turbulent World*, Emerald Publishing Limited, 2017, pp. 81–105. DOI: 10.1108/978-1-78714-242-820171005

[2] A. Nowacka and M. Rzemieniak, “The Impact of the VUCA Environment on the Digital Competences of Managers in the Power Industry,” *Energies*, vol. 15, no. 1, p. 185, Dec. 2021. DOI: 10.3390/en15010185

[3] E. Šimková and M. Hoffmannová, “Impact of VUCA Environment in Practice of Rural Tourism,” 2021, pp. 746–757. DOI: 10.36689/uhk/hed/2021-01-074

[4] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” *IEEE Software*, vol. 33, no. 3, pp. 42–52, May 2016. DOI: 10.1109/MS.2016.64

[5] M. Waseem, P. Liang, and M. Shahin, “A Systematic Mapping Study on Microservices Architecture in DevOps,” *Journal of Systems and Software*, vol. 170, p. 110798, Dec. 2020. DOI: 10.1016/j.jss.2020.110798

[6] P. di Francesco, P. Lago, and I. Malavolta, “Architecting with microservices: A systematic mapping study,” *Journal of Systems and Software*, vol. 150, pp. 77–97, 2019. DOI: 10.1016/j.jss.2019.01.001

[7] A. Rahmatulloh, D. W. Sari, R. N. Shofa, and I. Darmawan, “Microservices-based IoT Monitoring Application with a Domain-driven Design Approach,” in *2021 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS)*, 2021, pp. 1–8. DOI: 10.1109/ICADEIS52521.2021.9701966

[8] L. Sun, Y. Li, and R. A. Memon, “An open IoT framework based on microservices architecture,” *China Communications*, vol. 14, no. 2, pp. 154–162, 2017. DOI: 10.1109/CC.2017.7868163

[9] S. Trilles, A. González-Pérez, and J. Huerta, “An IoT platform based on microservices and serverless paradigms for smart farming purposes,” *Sensors (Switzerland)*, vol. 20, no. 8, 2020. DOI: 10.3390/s20082418

[10] E. K. Kannedy, “Event-Driven Architecture,” *medium.com*, 2018. [Online]. Available: https://medium.com/bliblidotcom-techblog/event-driven-architecture-ef3a312180ee

[11] S. Li *et al.*, “Understanding and addressing quality attributes of microservices architecture: A Systematic literature review,” *Information and Software Technology*, vol. 131, p. 106449, 2021. DOI: 10.1016/j.infsof.2020.106449

[12] D. Trihinas and G. Pallis, “DevOps as a Service : Pushing the Boundaries of Microservice Adoption Taking the Pulse of DevOps in the Cloud,” *IEEE Computer Society*, no. June, pp. 65–71, 2018.

[13] H. Kang, M. Le, and S. Tao, “Container and microservice driven design for cloud infrastructure DevOps,” *Proceedings - 2016 IEEE International Conference on Cloud Engineering, IC2E 2016: Co-located with the 1st IEEE International Conference on Internet-of-Things Design and Implementation, IoTDI 2016*, pp. 202–211, 2016. DOI: 10.1109/IC2E.2016.26

[14] X. J. Hong, H. Sik Yang, and Y. H. Kim, “Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application,” *9th International Conference on Information and Communication Technology Convergence: ICT Convergence Powered by Smart Intelligence, ICTC 2018*, pp. 257–259, 2018. DOI: 10.1109/ICTC.2018.8539409

[15] A. Akbulut and H. G. Perros, “Performance Analysis of Microservice Design Patterns,” *IEEE Internet Computing*, vol. 23, no. 6, pp. 19–27, 2019. DOI: 10.1109/MIC.2019.2951094

[16] R. A. Putra, “Analisa Implementasi Arsitektur Microservoces Berbasis Kontainer Pada Komunitas Pengembang Perangkat Lunak Sumber Terbuka (OpenDayLight DevOps Community),” *Jurnal Sistem Infomasi Teknologi Informasi dan Komputer (Just It) Universitas Bina Nusantara Magister Manajemen Sistem Informasi Jakarta*, pp. 150–162, 2018.

[17] L. Baresi, M. Garriga, and A. De Renzis, “Microservices identification through interface analysis,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10465 LNCS, pp. 19–33, 2017. DOI: 10.1007/978-3-319-67262-5_2

[18] J. Soldani, D. A. Tamburri, and W. J. Van Den Heuvel, “The pains and gains of microservices: A Systematic grey literature review,” *Journal of Systems and Software*, vol. 146, pp. 215–232, 2018. DOI: 10.1016/j.jss.2018.09.082

[19] M. Fihri, R. M. Negara, and D. D. Sanjoyo, “Implementasi & Analisis Performansi Layanan Web Pada Platform Berbasis Docker Implementation & Analysis of Web Service Performance Based on Docker Platform,” vol. 6, no. 2, pp. 3996–4001, 2019.

[20] L. P. Dewi, A. Noertjahyana, H. N. Palit, and K. Yedutun, “Server Scalability Using Kubernetes,” *TIMES-iCON 2019 - 2019 4th Technology Innovation Management and Engineering Science International Conference*, pp. 1–4, 2019. DOI: 10.1109/TIMES-iCON47539.2019.9024501

[21] H. Khazaei, C. Barna, N. Beigi-Mohammadi, and M. Litoiu, “Efficiency analysis of provisioning microservices,” *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 0, pp. 261–268, 2016. DOI: 10.1109/CloudCom.2016.0051

[22] M. Plauth, L. Feinbube, and A. Polze, “A performance survey of lightweight virtualization techniques,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10465 LNCS, pp. 34–48, 2017. DOI: 10.1007/978-3-319-67262-5_3

[23] T. Menouer, “KCSS: Kubernetes container scheduling strategy,” *Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, 2021. DOI: 10.1007/s11227-020-03427-3

[24] K. P. Singh, “Easy Kubernetes development with Skaffold,” *dev.to*, 2020. [Online]. Available: https://dev.to/karanpratapsingh/easy-kubernetes-development-with-skaffold-2ic8

[25] G. Jansen and J. Saladas, “Advantages of event-driven architecture,” *Developer IBM*, 2020. [Online]. Available: https://developer.ibm.com/technologies/messaging/articles/advantages-of-an-event-driven-architecture/

[26] S. Tragatschnig, S. Stevanetic, and U. Zdun, “Supporting the evolution of event-driven service-oriented architectures using change patterns,” *Information and Software Technology*, vol. 100, no. March, pp. 133–146, 2018. DOI: 10.1016/j.infsof.2018.04.005

[27] M. Khurma, “Percentage Increase - Formula, Examples, Application, Meaning.” [Online]. Available: https://www.cuemath.com/percentage-increase-formula/