Paper No: SE-01 **Systems Engineering**

An exploratory evaluation of replacing ESB with microservices in service-oriented architecture

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

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

***Abstract* - With the continuous progress in technology during the past few decades, cloud computing has become a fast-growing technology in the world, making computerized systems widespread. The emergence of Cloud Computing has evolved towards microservice concepts, which are highly demanded by corporates for enterprise application level. Most enterprise applications have moved away from traditional unified models of software programs like monolithic architecture and traditional SOA architecture to microservice architecture to ensure better scalability, lesser investment in hardware, and high performance. The monolithic architecture is designed in a manner that all the components and the modules are packed together and deployed on a single binary. However, in the microservice architecture, components are developed as small services so that horizontally and vertically scaling is made easier in comparison to monolith or SOA architecture. SOA and monolithic architecture are at a disadvantage compared to Microservice architecture, as they require colossal hardware specifications to scale the software. In general terms, the system performance of these architectures can be measured considering different aspects such as system capacity, throughput, and latency. This research focuses on how scalability and performance software quality attributes behave when converting the SOA system to microservice architecture. Experimental results have shown that microservice architecture can bring more scalability with a minimum cost generation. Nevertheless, specific gaps in performance are identified in the perspective of the final user experiences due to the interservice communication in the microservice architecture in a distributed environment.**

Keywords - microservice, performance, scalability, SOA

I. INTRODUCTION

Since the world is more inclined towards new technology, it has ultimately resulted in an information system-driven society. People are concerned about attending to their routine tasks in the most efficient, easy, and fastest method possible. Because of this driving need to achieve efficiency and effectiveness, the necessity to successfully build systems to win over these real-world problems was considered vital by software engineers. Researching and proposing new software architectural concepts by the software industry were initiated to develop the most reliable software in the world [1]. These architectures give a better view of the software to provide the services and evolve the quality of its life cycle. Architecture is responsible for providing the bridge for the software functionalities and the system quality attributes necessary for the business needs. As a first step, the engineers develop object-oriented architecture patterns that cater to the small-scale software run on the host machines.

Historically, the software industry developed monolithic software for enterprise-level solutions. The traditional monolithic application encapsulates all the components, functions into one single package and deploys as a single application. Most of the service-oriented monolithic applications are developed using the C, C++, Java, and Python languages. Those languages by default support creating the single executable artifact. Some of the monolithic systems are deployed in the distributed environment using the RMI, Network Object, and CORBA concepts. However, it's tough to maintain the monolithic in the distributed environment [2].

On the contrary, there are many advantages of using the monolithic systems such as easy deployment because all the modules are in the same code base, supportive nature of the entire IDEs, ease of testing the entire system as there's no requirement to set up various components, and the ease of scaling since monolithic application comes up with the option of a single distribution. However, the monolithic application has significant drawbacks, which are mostly related to business growth and technology adaptations. For instance, all the components are packed together in monolith architecture with a vast codebase; hence, it's complicated to make modifications. Also, the application patching process and understanding the monolithic applications are quite challenging. On the other hand, one single failure of the application can cause the collapse of the entire system. Therefore, it can be derived that those monolithic applications are not suitable for deployment in the containerization environment. Monolith applications are cumbersome, and it takes a considerable amount of time to startup. Continuous integration and continuous delivery pipeline are complicated to maintain with monolithic systems because of the heaviness of the systems. One single change needs to test the overall system functionalities as of the tightly coupled components. Hence overall time to test and the cost generated for deployment will be considerably high.

With the concept of the "separation of concerns," component-based software engineering comes into the world, which leads to better implementation, design, and evolution of software systems. Then the Service-Oriented Computing (SOC) paradigm comes into context. People moved to distributed software development and deployed that software in the distributed environment [3]. In SOC, each component's functionalities are shared using the message passing through those distributed components. The SOC architectural concept brings several advantages to the software industry, such as "dynamism**"** which can introduce the same component based on the system load, modularity which can be reused across the components, and distributed development.

In the mid-'90s, Gartner Group researchers introduced a reference architecture for the industry called service-oriented architecture (SOA) [4]. In SOA architecture, both the service consumers and service providers get together

and provide the business needs. Services are the distributed components, and they have published the interfaces to do the communication via middleware. Those interfaces abstract all business logic. One of the main components of service-oriented architecture is the enterprise service bus (ESB) which serves as middleware. ESB's main task is to enable communication between those services and govern them. Most of the SOA systems use the Simple Object Access Protocol (SOAP) for communication. SOA architecture data sources are shared with the components deployed in the same environment. That means the same database is open for both Data Definition Language (DDL) and Data Manipulation Language (DML) and all the components residing inside the SOA architecture.

The difference between SOA and monolithic architecture is that SOA architecture consists of the component as a service, but the monolithic builds all the logic in one package. In the monolithic architecture, all the logic is based on sharing one single hardware resource. Nevertheless, in SOA architectures, each component uses its hardware resources to provide the service. Compared to the monolith applications, SOA brings more advantages to the software industry, such as enabling the system's growth to the enterprise level, bringing component-wise scalability to the whole environment, and reducing operational costs.

The term "Microservice" was initially introduced in 2011 at an architectural workshop conference [2]. Microservice architecture comes into the world as a new architectural paradigm that can be illustrated as tiny services running independently and communicating with each other and satisfying the business requirement. The microservice architecture was widely used by people in the past few years, which can be considered as a positive behavior to the software industry. With time, most software firms arrived at the notion that using the microservice architecture developments brings high productivity to the company and produces a successful end product for the clients [5]. Microservice architecture also takes advantage of cloud services such as on-demand provisioning, serverless functions, and elasticity as well as a lot of quality attributes such as scalability, maintainability, performance and many more.

People who intend to move away from the monolithic to SOA architecture should particularly comprehend the quality attributes generated by it. In this paper, our acute concentration is on evaluating and coming up with the architectural conclusion on the extremely critical quality attributes which diverge from the most common SOA architecture with ESB and the Microservice architecture.

II. Background and Related Work

Microservice architecture is derived from the concept of the SOA. Microservices are now considered the new software architecture for highly scalable and highly maintainable distributed systems. Nevertheless, when the system functionalities grow day by day, microservice architecture tends to get complex because of the large set of independent services it has as functions. Developing and deploying the microservices independently to each other brings high cohesion and loosely coupled modules [6].

The reason behind the popularity of the microservices architecture is the quality attributes associated with the microservices. We identified the most concerning quality attributes on the microservices architecture, such as scalability, performance, availability, maintainability, and security [7, 8].

A. Quality attributes in microservice architecture

Several definitions can define "Quality" in a microservice architecture. Some people denote it by the software's capability to meet the required requirements, and some of the people define it as the "reality of the objectives" [9]. In the context of software engineering, quality refers to the relationship between the business and the product. This software quality contains two types;

Software functional quality – Describes the functional requirements with the current system design. Functional quality attributes show how the system matches the business requirement. Using this quality, people can decide whether the developed software is acceptable or not.

Software structural quality – Describes the software non-functional requirements that support in providing the functional requirement on the system. Those non-functional requirements bring more value addition to the software ecosystem.

The software stakeholders are primarily concerned about the system requirements. Based on the stakeholder requirements, we can divide software quality into two main groups; the development phase and the operations phase. In the development phases, we need quality requirements that are very important for software developers, such as maintainability, modularity, and understandability. Quality requirements for the operations related to the system end-users and system supporting teams include usability, traceability, availability, and performance.

Those quality requirements have differed from the software domain, priorities of the developers, and the end-users. We can see the quality attributes when the system has been implemented.

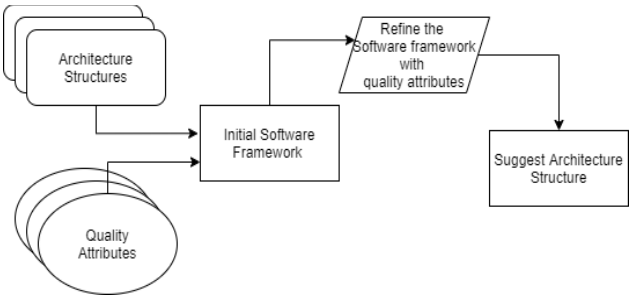

Fig. 1. How quality attributes influence to software architecture

According to Figure 1, all the quality attributes are depending on the software architecture [10]. It is mandatory to review the software architecture before the software development or use the reference architecture to develop the software. The qualities cannot be added to the system architecture ad-hoc. Therefore, developers need to build those qualities from scratch on the software.

B. Scalability

The scalability quality attribute is one of the primary critical features in the microservice architecture. The scalability attribute was initially introduced to enhance software performance and control high traffic. Scalability

quality also ensures the system fault tolerance. There are two main parts of scaling.

Horizontally Scaling- This method ensures that the application's performance is increased by adding another application instance over it. For example, we have one web server before scaling, and after scaling, we have multiple web servers that serve traffic. Load balancers help to distribute the traffic load among those web servers [11].

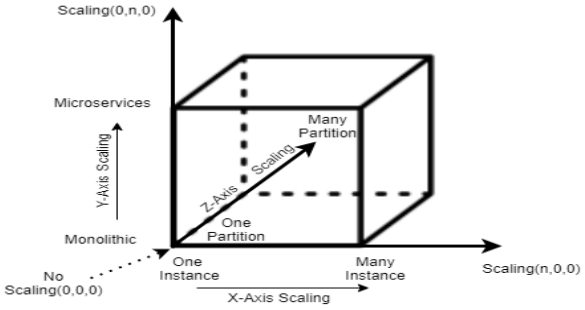

Fig. 2. Scaling cube

Vertically Scaling- This means increasing the hardware resource to improve the application performance, such as increasing the RAM, increasing the CPU, and using the SSD instead of HDD storage [12]. Vertical scaling is a very traditional method, and most people use computers to do this kind of scaling. For instance, vertical scaling is majorly used when the personal computer is slow and the need to increase the computer hardware occurs. Nevertheless, this scaling is bound to a limited area, and there's no possible way to increase the hardware resource as we want. Because particular hardware only supports the specific ranges only. As an example, some motherboards' maximum supported RAM is 64GB.

Scaling cube shows scaling model for the software applications [13]. We also refer to this concept when scaling the application in our research. Figure 2 X-Axis scaling is referred to as horizontally scaling, work evenly distributed scaling, and horizontally duplication. The simple meaning is that running the software application behind the load balancer. The Load balancer is responsible for the equal distribution of the load among the number of applications connected to the load balancer rules. X-Axis scaling is mostly used by monolithic applications with shared databases and caches.

Y-Axis scaling applications are decomposed to the small binaries by considering the functions/services called microservices. (0,0) indicates the monolithic application, which contains all the services as one single binary. Y-Axis scaling gives more value to the software architecture because services behave independently. Therefore, people can only scale the relevant services using this concept.

The microservice architecture is a combination of both X and Y-Axis scaling. This helps bring more scalable software architecture to the deployments.

Z-Axis scaling is somewhat similar to the X-Axis scaling, but it differs from the data used by the application. For instance, assume that we have a significant number of students, and according to the admission number, they are segregated into groups. In each group, the same application is running and doing the same service but using different data. This is primarily applicable to B2C applications. The load balancer should need to be intelligent to recognize the correct data partition server to route the traffic. Otherwise, we need to put the router before those servers.

When it comes to a cloud-native architecture, most cloud providers such as Amazon Web Services (AWS), Google Cloud Service (GCP), and Azure develop various vertical and horizontal scaling solutions. Most prominent players, such as Netflix, Uber, WhatsApp, and Instagram, also deploy their applications in cloud-native environments [11]. Using the virtualization technology, the cloud providers introduce vertical and horizontal scaling on the cloud resources such as servers, storage, and databases. They have introduced AI technologies like machine learning to perform predictive analysis on the scaling part and automatic scaling. Day by day, those reactive scalings become seamless with the help of those AI technologies. Most of the cloud-native applications developed as containerized applications and deployed on container orchestration engines like Kubernetes. Cloud providers also give services to cloud consumers by enabling the container orchestration engine. For example, the AWS cloud provider gives Amazon Elastic Container Service (Amazon ECS) and Google cloud to provide the Google Kubernetes Engine (GKE). Those services will take care of managing the whole container orchestration part. The developer needs only to develop the application which is suitable for cloud-native environments. In this cloud-native environment, containers are warped as small pods that allow the scaling up and down in a simple way.

C. Performance

Performance is one of the most critical quality attributes. Both software consumers and the developer care about application performance during the run time. Performance is measured by the measurable factor of the system when performing the given functionalities within given constraints such as accuracy, latency, and resource consumption. A simple way to define the performance in the software is how software behaves on time, which is called responsiveness [12]. Most people move away from manual work to digitalized platforms with the belief that such work can be done in lesser time and minimum effort. The outcome of the software system should always be; consumption of less amount of time with more accuracy. The main objective of the real-time system is to give a response in real-time. For that, system architectures and software design also need to be well established. In the past decade, most of the performance issues were identified in the production environments since unpredictable behaviors of the users who are using the software and the unpredictable behaviors in the environment are found to be the root causes for performance issues. To reduce the above issue, the performance factor is considered when the system is in the design phase.

There are several criteria to check the performance of the software system.

a) Latency / response time

This refers to how much time is taken to complete the task and respond. If the time difference between start time and end time is low, that means the system performance is good. API-based synchronized system's API response time measure using the microseconds and milliseconds.

b) Throughput

Throughput refers to the number of tasks that have been completed within the given time interval. In other words, it is the software process rate or the time frame as seconds. It's also called transactions per second (TPS). Measurement of the throughput is different from application to application. High throughput means software performance is in a good state.

c) Capacity

This means how much work software can perform. The maximum throughput is considered as system capacity. In other words, the maximum number of events the software can perform within a unit of time and total resource consumption. For example, software A can support a maximum of 250 TPS with 1s latency backend AWS m4.large VM (8GB RAM, 2vCPU) and network perspective bandwidth means the capacity. When the capacity is getting immense value, then we can consider that the software performance is high.

III. Research Methodology

This research will talk about the most concerning quality attribute variation when converting software architecture from SOA to microservice architecture. By critically reviewing the software architecture, we identified that scalability and performance are the most critical quality attributes in the software industry [8][9]. After the monolithic architecture, software architects introduced the SOA. However, we can identify some limitations on the scaling and the performance quality attributes by reviewing the SOA. There were several problems identified when scaling the SOA-based system. All the services are decoupled in the SOA-based system and exchange the required data via the enterprise service bus (ESB). ESB is responsible for the service orchestration, and it acts as a backbone of the SOA system. When scaling the SOA-based system, at one point, people need to scale the ESB also. So scaling ESB requires high-end specification servers that will generate a considerable amount of cost. ESB servers contain many features and modules, and in some cases, the software ecosystem did not use all of the features carried on the ESB servers in SOA. Because of that, performance-wise, it has some impact on the SOA systems during run time. With those factors, people are moving from Software Oriented Architecture to microservice-based architecture. This research evaluates how scalability and the performance quality attributes vary when transforming SOA to the microservice-based architecture.

We have developed the SOA system that can talk with the legacy backend, and at the same time, we have developed business functionalities using microservice-based architecture, which can also communicate with the legacy backends.

Fig. 3., shows how the SOA system integrates with the databases, backend, and clients. ESB is responsible for catering the message routing and publishing all the communication to the data source.

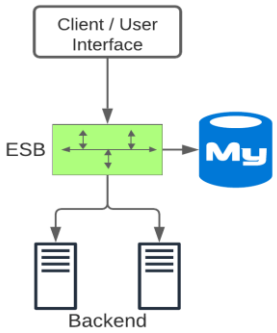

Fig. 3. SOA architecture

Here we use the WSO2 Enterprise Service Bus, an open-source product, and most of the well-known software companies use this product for their software systems as well [14]. We choose WSO2 ESB as it generates many features like better performance and user-friendly nature compared to other ESBs. Also, in WSO2 ESB, the lightweight mechanism is introduced, and also it is an open-source product [14] [15]. With the WSO2 ESB, we wrote the business logic using the Apache Synapse language [17] and deployed it as Carbon applications in the ESB servers [18]. All the products of WSO2 are based on the Carbon platform. This is a form of middleware platform that stores business IT projects on the cloud, and on-premises servers [19]. With the help of the WSO2 developer studio, WSO2 ESB has created the opportunity for the software developers to swiftly orchestrate applications, business processes, and the services such as data service, proxy-based service, message routing service, etc. With this kind of development, software companies can deliver the services promptly to the clients. Moreover, the technical and the business services can be integrated with the legacy systems and any kind of SAAS services in SOA architecture. Backend is a legacy that one can communicate using the REST protocol. Clients/User interface communicates to the ESB using the REST protocol by exposed APIs.

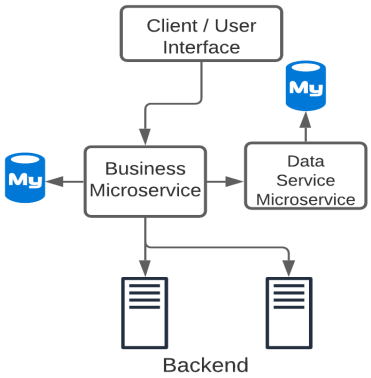

Fig. 4. Microservice architecture

Figure 4 shows how the microservices replace the SOA System. We have identified the ESB server's required services and made those services into individual components and deployed them as microservices. Business microservice consists of all business logic, and data service

is responsible for publishing data. Here we used the same legacy backend, which can communicate with the REST protocol with the microservices. This microservices architecture is developed using JAVA language with the help of the Spring boot framework. REST client libraries are used for inter-service communication with the microservice to microservice and other services. Business logic microservice has exposed the APIs using the request controllers to communicate with the clients/user interfaces.

IV. Results and Evaluation

The developed two systems were evaluated in the real environment with two main quality attributes: performance and scalability. In scalability, we are more concerned about the hardware footprint and the cost. There are several aspects of performance. In this, we evaluated the latency, throughput, and capacity with the allocated hardware. Throughout the experimental time, we collected statistics about the load average of the server, memory usage on the server, overall response time of the application, and throughput of the application using the JMeter [20]. Applications' ramp-up time frame and the steady-state time frame are included in those statistics. Firstly, we hosted the application in the different servers which are having different footprints. Then we collected the above statistics in those different environments by sending the 1KB size POST JSON payload to the applications. Upon collecting the statistics and sending the payload, backend servers returned the 1KB size JSON response. We use the Amazon Web Services (AWS) environment for all the environments. As a client, we used Apache open source JMeter [20] to generate the load toward the deployed servers. For all stress tests, we used 350 concurrent threads. In the AWS environment, T2 type resources were used in our experiment because of the following several reasons: It has Intel Xeon processors with high frequency that can be burstable, its coherent baseline performance is suitable for the general-purpose application deployments [21], and It is capable of balancing the overall server resources (CPU/memory/network).

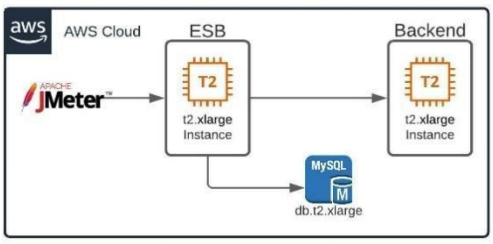

Fig. 5. 1st Test suite architecture

As the first test suite shows in Figure 5, we used the AWS t2.xlarge EC2 instance with four virtual CPUs and 16GB RAM. Also, the Solid-State Drive (SSD) was used to store the application. Then we deployed the WSO2 ESB application with customized development using the synapse language to cater to business logic. The ESB server connects with the AWS RDS MYSQL database service, which is deployed in the same VPC to reduce network latency. We used db.t2.xlarge, which has four virtual CPUs and 16GB RAM. Simultaneously, we provisioned the 100GB storage size for this RDS.

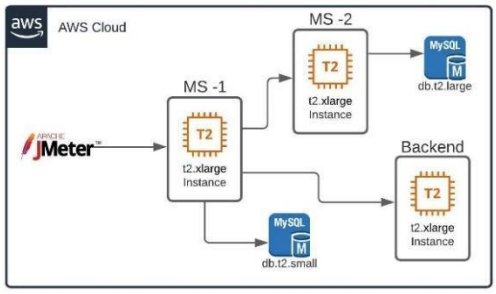

Fig. 6. 2nd Test suite architecture

The microservices for the second test suite, as shown in Fig. 6, that can perform the same ESB business logic relevant to this deployment, was developed. It had two microservices, and those microservices are deployed in the AWS t2.xlarge EC2 instances with Solid-State Drive (SSD) storage. Following the microservice concept, two different databases which are deployed in the same internal network. db.t2.large type RDS with 100GB storage was used for the data service microservice, and db.t2.small type RDS with 20GB storage was used for business microservice.

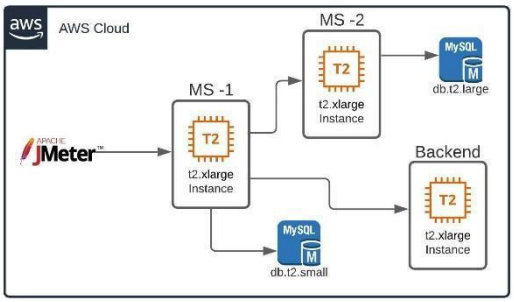

Fig. 7. 3rd Test suite architecture

For the third test scenario shown in Fig. 7, we reduced the server footprint after analyzing the statistics we collected on the 2nd test suite. For both the microservice deployments, we used the t2.medium AWS EC2 instances, which have 2 virtual CPUs and 4GB RAM. We used the Solid-State Drive (SSD) in both servers to store the application. The same database type was used in the 2nd test suite without any modifications. All the servers and the database were placed in the same internal network.

The backend servers and the client server (JMeter) were not changed for any of the testing scenarios. For storage, AWS t2.xlarge EC2 instances with Solid-State Drive (SSD) were used for both backend servers and the client servers. These two servers were also placed in the same internal network as the other servers.

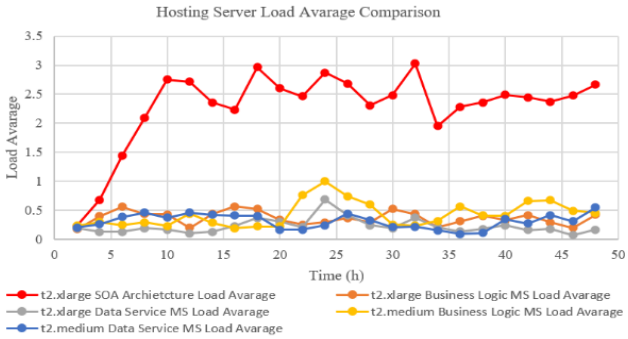

Fig. 8. Load average comparison

Figure 8 shows how the average server load varies on the SOA architecture and microservice architecture systems on the different hardware footprints. In the SOA architecture, the ESB node consumes many load averages to process the client requirement. However, none of the microservices deployed in the two different server types went for more than one load average.

If we group and add up the t2.xlarge two microservices load averages, those added up values will not be higher than the SOA architecture load average values. This is the same for the t2.medium microservices load average as well. It was found that Microservice architecture deployment was able to work with less resource consumption once we were vertically scaled-down the servers. On the contrary, ESB servers could not vertically scale down because they have fully utilized the current server resources.

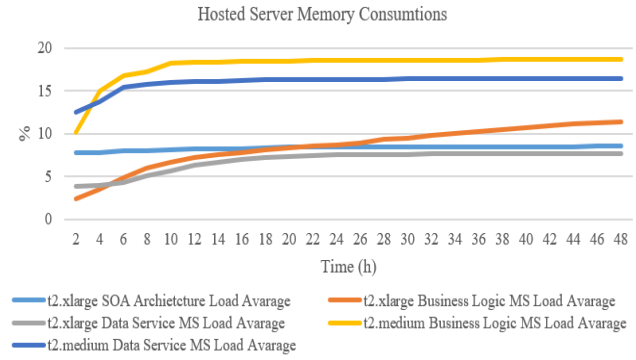

Fig. 9. Memory usage comparison

Fig. 9 shows the memory consumption on the SOA architecture system and the microservice architecture systems. None of the servers consume the 20% server RAM. When vertically scaling down the microservices, it We can see a slight improvement in the throughput in figure 11 when vertically scaling the hardware footprint in the microservice architecture was observed that it increases the memory by nearly 5% on both the data service microservice and the business logic microservice.

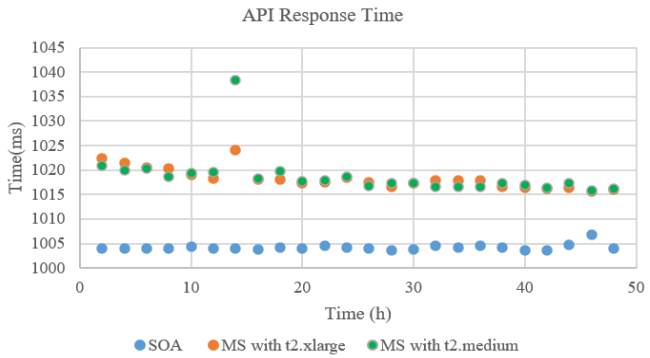

Fig. 10. Response time variation

Fig.10 shows the overall response time on each system with the deployed environment. SOA system performs with less response time in comparison to the microservice architecture system. It does not deviate much from the environment, and its software architecture. In the SOA system, all the modules we packed in the ESB server and no network calls for satisfy the full business function. All the logic is handled inside the single JVM. Because of that, response time is lower than the microservice architecture. The reason behind having a higher response time in the microservice architecture is because of the network call to the separate services. It introduces the additional time for the overall response time.

SOA system shows high performance by producing within a less response time. However, system throughput is less than the microservice. At a single time, the slice system only handles the smaller number of concurrent requests rather than the microservices. Because the SOA system consists of all the modules in the same JVM, and it takes all the resources on the JVM. So, the server does not accept the high number of requests to the single run time environment.

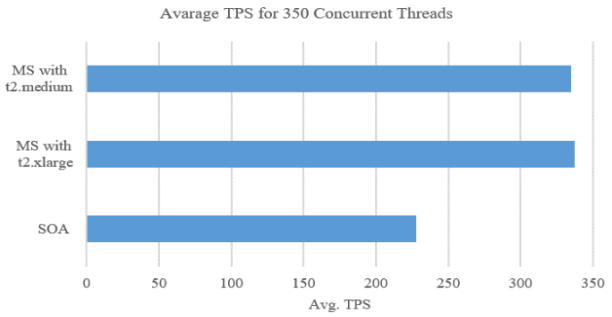

Fig. 11. Throughput comparison

.

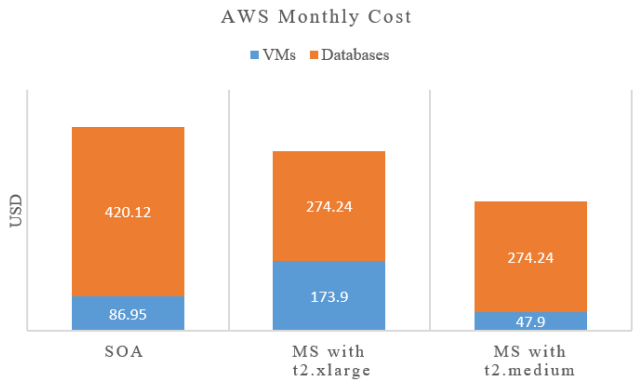

Fig. 12. Cost comparison

Fig. 12 graph only considers the dynamic values we have used in different test suites. Comparing the cost of both SOA and the microservice architecture shows that SOA generates a higher cost for the entire end-to-end deployments [22]. Experienced system architects can determine the exact footprint for the developed microservice by considering the user requirements. Using the optimal hardware footprint, we can save much money on software deployment projects. Those microservice can deploy the Kubernetes environments without putting more effort. From that, we can do the auto-scaling as per the traffic load. With this also we can save the overall cost.

V. Conclusion and Further Work

This topic unfolds the factors to evaluate the research problem, which is the most concerning quality attributes of scalability and the performance variate between the Service Oriented Architecture and microservice architecture. Most organizations expect microservice architecture to move their current monolithic architecture or SOA. The main concern with the current monolithic and SOA architecture is the cost of scalability. Their current deployment footprint is also high, and it already involves a considerable cost. When we were going to scale that current environment, it made the cost nearly double. Nowadays, all the systems are deployed as contained in a cloud-native environment.

Nevertheless, monolithic and SOA-based architecture systems are not suitable for cloud-native environments. Because those applications are enormous and take a considerable amount of time to startup and serve the traffic, if we put those kinds of applications in the Kubernetes environments as pods, we cannot get the advantages provided by the container orchestration engines. Nevertheless, when converting to cloud-native microservices, some of the performance factors get affected. Before converting the monolithic / SOA system, we need to think about what performance factor requires enhancement. In terms of capacity and cost-effectiveness, microservice could be considered a better approach. When we move to the microservice architecture, we have flexible scalability. Through Microservice architecture, people have the option of only scaling the necessary services rather than the entire application. The previous chapter shows the fundamental analysis, and this could assist researchers in concluding microservice architecture.

In summary, we could state that microservice architecture is a better approach in terms of scalability and performance in comparison to SOA and monolithic architecture. The research study results clearly showed that microservice architecture gives more performance in terms of the throughput and the application's capacity. Moreover, it is a cost-effective solution when scaling the applications. With this study, architects can redesign existing microservice architecture applications and adhere to cloud-native environments. Future work needs to find a solution for reducing the performance impact on latency in the microservice architecture.

References

[1] R. Flygare and A. Holmqvist, "Performance characteristics between monolithic and microservice-based systems," Blekinge Inst. Technol., 2017.

[2] N. Dragoni et al., "Microservices: Yesterday, Today, and Tomorrow," in Present and Ulterior Software Engineering, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216. doi: 10.1007/978-3-319-67425-4_12.

[3] MacKenzie, K. Laskey, F. Mccabe, P. Brown, and R. Metz, "Reference model for service oriented architecture 1.0," Public Rev Draft, vol. 2, pp. 1–31, Aug. 2006.

[4] R. Mohan, T. Ramanathan, G. Rajendran, and D. N. MohanRaj, "Gartner Research Reviews on Middleware," Int. J. Sci. Res. Publ., vol. 5, no. 9, p. 2, 2015.

[5] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposl and a discussion," Requir. Eng., vol. 11, no. 1, pp. 102–107, Mar. 2006, doi: 10.1007/s00766-005-0021-6.

[6] N. Alshuqayran, N. Ali, and R. Evans, "A Systematic Mapping Study in Microservice Architecture," in 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, Nov. 2016, pp. 44–51. doi: 10.1109/SOCA.2016.15.

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

[8] S. Li et al., "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," Inf. Softw. Technol., vol. 131, p. 106449, Mar. 2021, doi: 10.1016/j.infsof.2020.106449.

[9] A. Chandrasekar, M. SudhaRajesh, and M. P. Rajesh, "A Research Study on Software Quality Attributes," Int. J. Sci. Res. Publ., vol. 4, no. 1, p. 4, 2014.

[10] M. Svahnberg, C. Wohlin, L. Lundberg, and M. Mattsson, "A Method for Understanding Quality Attributes in Software Architecture Structures," in Proceedings of the 14th international conference on Software engineering and knowledge engineering, Jan. 2002, p. 8. doi: 10.1145/568760.568900.

[11] N. Kratzke, "A Brief History of Cloud Application Architectures," Appl. Sci., vol. 8, no. 8, p. 1368, Aug. 2018, doi: 10.3390/app8081368.

[12] U. Smith and L. G. Williams, "Software performance engineering: a case study including performance comparison with design alternatives," IEEE Trans. Softw. Eng., vol. 19, no. 7, pp. 720–741, Jul. 1993, doi: 10.1109/32.238572.

[13] Marquez, M. M. Villegas, and H. Astudillo, "An Empirical Study of Scalability Frameworks in Open Source Microservices-based Systems," in 2018 37th International Conference of the Chilean Computer Science Society (SCCC), Santiago, Chile, Nov. 2018, pp. 1–8. doi: 10.1109/SCCC.2018.8705256.

[14] S. Sasono, F. R. Rumambi, R. Priskila, and D. B. Setyohadi, "Integration of pharmacy and drug manufacturers in RSUD Dr Samratulangi Tondano by ESB WSO2 to improve service quality: (A case study of RSUD Dr Samratulangi Tondano, Minahasa Regency, North Sulawesi)," in 2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Semarang, Oct. 2017, pp. 249–254. doi: 10.1109/ICITACEE.2017.8257712.

[15] sofwarewiki, "7 Excellent Open Source ESB (Enterprise Service Bus) Alternatives." https://www.fromdev.com/2012/03/7-excellent-open-source-enterprise.html

[16] Chanaka Fernando, "Five Reasons Why WSO2 is Better Than Mule." WSO2, Sep. 23, 2020. [Online]. Available: https://wso2.com/blogs/thesource/five-reasons-why-wso2-is-better-than-mule/

[17] J. Ma, H. Yu, and J. Guo, "Research and Implement on Application Integration Based on the Apache Synapse ESB platform," AASRI Procedia, vol. 1, pp. 82–86, 2012, doi: 10.1016/j.aasri.2012.06.015.
[18] WSO2, "Quick Start Guide - Enterprise Service Bus 5.0.0 - WSO2 Documentation." WSO2 Inc. [Online]. Available: https://docs.wso2.com/display/ESB500/Quick+Start+Guide
[19] J. Krein, "Web-based application integration: advanced business process monitoring in WSO2 carbon," 2011, doi: 10.18419/opus-2719.
[20] R. B. Khan, "Comparative Study of Performance Testing Tools: Apache JMeter and HP LoadRunner," p. 57.
[21] "Amazon EC2 Instance Types - Amazon Web Services," Amazon Web Services, Inc. https://aws.amazon.com /ec2/instance-types/
[22] "AWS Pricing Calculator." https://calculator.aws