Performance and complexity comparison of service oriented architecture and microservices architecture

Vinay Raj* and Ravichandra Sadam

Department of Computer Science and Engineering,
National Institute of Technology Warangal,
Telangana, India
Email: rvinay1@student.nitw.ac.in
Email: ravic@nitw.ac.in
*Corresponding author

Abstract: Microservices has emerged as a new architectural style of designing applications to overcome the challenges of service oriented architecture (SOA). With the evolution of microservices architecture, architects have started migrating legacy applications to microservices. However, some of the architects are in chaos whether to migrate the application from SOA to microservices or not. The need for empirical evaluation and comparison of both the SOA and microservices architecture is also on-demand. This work helps the software architects in better understanding of the technical differences between both styles. We, therefore, present a comparison of a web application that is designed using both SOA and microservices architectures. The comparison is presented with two different parameters: 1) *complexity* with architectural metrics; 2) *performance* with load testing. It is clear from the results that though the complexity of microservices architecture is high, the response time for processing the requests is very fast compared to SOA services.

Keywords: service oriented architecture; SOA; microservices; architectural metrics; performance testing; complexity; coupling.

Reference to this paper should be made as follows: Raj, V. and Sadam, R. (2021) 'Performance and complexity comparison of service oriented architecture and microservices architecture', *Int. J. Communication Networks and Distributed Systems*, Vol. 27, No. 1, pp.100–117.

Biographical notes: Vinay Raj received his Master's in Software Systems from the BITS-Pilani, India in 2016. He is currently a PhD student at the Computer Science and Engineering Department of the National Institute of Technology Warangal (NITW), India. He has worked for TCS and Capgemini, India as a software developer in SOA projects. His research interests include service oriented architecture, microservices and software engineering.

Ravichandra Sadam is an Associate Professor in the Department of Computer Science and Engineering, National Institute of Technology (NIT) Warangal, Telangana, India. His research interests include software engineering, software architecture, design patterns, and service-oriented architecture. He is a member of IEEE and ISTE.

1 Introduction

Distributed systems have evolved rapidly as the demand for quick design and deployment of business requirements has increased (Salah et al., 2016). They have grown rapidly from the monolithic style of designing applications to today's service-based architectures such as service oriented architecture (SOA) and microservices. In monolithic applications, the entire application is dumped in a single code base and deployed as a single archive. Monolithic applications are difficult to update, maintain, and deploy as it makes the application code very complex to understand (Escobar et al., 2016). To make a single update in the system, the entire application needs to be shut down and redeploy every component (Krasuse, 2015). To overcome the design and deployment challenges in monolithic applications, SOA has emerged as a style of decomposing the entire application into loosely coupled, scalable, and interoperable services (Xiao et al., 2016). Services are the major component of the SOA that performs various business tasks that are simple or complex based on the business requirements. SOA is mainly used for the integration of large enterprise applications using middleware communication technologies such as enterprise service bus (ESB) (Raj and Ravichandra, 2018). SOA has gained more popularity with the web services implementation where each service can be designed, deployed, and discovered over the internet using the simple object access protocol (SOAP) over hypertext transfer protocol (HTTP) (Vural et al., 2017).

Though SOA has gained popularity in designing enterprise applications, it exhibits few design and deployment challenges (Salah et al., 2016). The services in SOA are tightly coupled with ESB for the exchange of messages and if there is a need to update a particular service, it requires to redeploy the dependent components in the system. Therefore, the deployment process of SOA is still seen as a monolith (Wolff, 2016). A lot of complex configurations in ESB are required for adding new requirements for end-users in SOA. Moreover, with the ever-changing business requirements, the services in SOA are tending towards monolithic in size making the application complex and difficult to maintain. Further, the on-demand services can be scaled horizontally but the hardware cost increases as it requires additional infrastructure. The web services implementation uses heavyweight protocols such as extensible markup language (XML)-based web services description language (WSDL) and SOAP for communication between the services which reduces the performance of the application.

To overcome the challenges in existing architectures, microservices has emerged as a new style and has become popular in the designing of enterprise applications. Microservices is defined as an architectural style for developing applications as a suite of small and independent components. Each microservice runs in its own process and communicates with other services using lightweight protocols such as representational state transfer (REST) and HTTP (Fowler and Lewis, 2014). Loose coupling and the freedom of choosing programming languages for the implementation of microservices is one of the major benefits. Microservices are deployed in docker containers which are lightweight and they are best suitable for microservices as they start very quickly (Amaral et al., 2015). Container images consist of all required environmental configurations and developers can easily access from DockerHub. Microservices can be easily added or removed from the applications and they can be easily migrated from one host to another. Independent deployment helps in auto-scaling of the microservices at a fast pace and can easily handle the load. Microservices enable continuous integration

and continuous delivery and suit well with DevOps style which acts as a framework for the complete software development life cycle (SDLC) of microservices (Balalaie et al., 2016). Compared to traditional architectures, microservices are designed in short development time, reduces the inherent complexity, and increases the scalability (Wang et al., 2020).

Microservices has become popular and many large information technology (IT) giants such as Amazon, Netflix, Twitter, etc. have started designing their applications using microservices. Many companies have also started migrating their legacy applications to microservices architectural style (Taibi et al., 2018). According to the International Data Corporation (IDC), 80% of the cloud applications will be designed using microservices by the end of 2021 (Larrucea et al., 2016). However, academic research of microservices is still at an early stage and very little work has been published on the comparison of microservices with SOA.

Technically, there are certain differences between the design and implementation strategies of both SOA and microservices applications. SOA is based on the concept of sharing as much as possible whereas microservices architecture is based on the idea of sharing as little as possible (Wilde et al., 2016). SOA depends on heavyweight middleware and ESB for communication between the services whereas microservices rely only on lightweight technologies. SOA is related to protocols and formats such as WSDL, SOAP, etc. and microservices use REST and HTTP for communication and JavaScript Object Notation (JSON) as data exchange format (Jamshidi et al., 2016). Microservices follow the concept of smart endpoints, dumb pipes, and follow the strategy of choreography over orchestration (Bogner et al., 2017).

1.1 Motivation

With the wider benefits of microservices, software architects have started pushing their applications towards microservices architectural style. However, with the costs required for migration, architects are hesitant to migrate and they are in chaos whether to migrate the applications to microservices or not as they are unaware of the pros and cons of adopting it (Taibi et al., 2017). Also, in several studies, it is mentioned that there is a need to compare and investigate both SOA and microservices architecture in terms of performance, development effort, and maintenance (Bogner et al., 2019; Rademacher et al., 2018; Pahl et al., 2018). Scalability of an application cannot be measured directly and it is an immeasurable unit. However, scalability depends on coupling and complexity of the services, which we have considered in this work. Therefore, this work presents the empirical comparison of both SOA and microservices architecture with a prime focus on the complexity and performance of the applications.

The remaining part of the paper is organised as follows. The required background information about complexity and performance is discussed in Section 2. Section 3 talks about the related works done in comparing both SOA and microservices architectures. The preliminaries of service graph (SG) concepts are discussed in Section 4 and the case study application along with its service details are presented in Section 5. The comparison in terms of complexity and performance are presented in Sections 6 and 7 respectively. Section 8 concludes the paper.

2 Background

One of the goals in the architecture-level modifiability analysis (ALMA) model presented by Lassing et al. (2002) considers comparing two or more architectures to find the better one. Here, we consider SOA and microservices architectures for comparison to find the appropriate one to use in the design of enterprise applications. The number of services (NoSs) will be more in microservices and the applications will be more complex compared to SOA. Moreover, the deployment strategy of microservices is completely different from that of SOA and hence it motivates us to select the comparison criteria as complexity analysis and performance testing of the service-based architectures.

2.1 Complexity

In the context of software architectures, complexity can be defined as the complexities of services or components that make up the complete architecture and their dependencies (AlSharif et al., 2004). Continuous upgrade and enhancements of software applications make the services larger and hence the design, implementation, and deployment of such applications become more complex (Gribaudo et al., 2017). The results of the survey presented in Di Francesco et al. (2017) state that despite having low service coupling in microservices, the complexity of the application increases. Microservices system consists of fine-grained services and interactions are very complex including the configurations of the environments. Multiple service calls occur for a simple business task to process and give the result (Zhou et al., 2018). Also, the cost and time for development increases with the increase in the complexity of the application (Fowler and Lewis, 2014). Though it has been stated in the literature that the complexity of the microservices application is high, it is required to compare the same with SOA and check the behaviour of the application with the increase in complexity.

2.2 Performance testing

Performance related features such as response time are of more interest to determine the acceptability of software design (Spitznagel and Garlan, 1998). Response time is the time taken for a particular business request (BR) from initiating to the successful completion of the task. Though the chosen architectures are service-based, the implementation style and deployment environment are completely different and the impact of cloud can be analysed with load testing. JMeter tool is best suitable for performing the load testing and it has been successfully used to evaluate the SOA-based web services (Nevedrov, 2006). In a comparative study (Salah et al., 2017), container-based services perform better when compared to virtual machine (VM)-based services. Hence, we deploy microservices in cloud containers and verify the response times of both applications.

3 Related work: SOA vs. microservices

The comparison of SOA and microservices have been studied in different perspectives in the literature. Bogner et al. (2018) have considered the relevance of SOA patterns to microservices architecture. The patterns used in microservices are not new and most of the patterns applicable for microservices are also applicable to microservices. However, this cannot be considered as a comparison of both the architectures. Theoretical differences between both the styles in terms of features are discussed in Cerny et al. (2017, 2018). The differences are presented in both research and industry perspective. Similar work of comparing distributed systems such as client/server, mobile agents, SOA, and microservices is presented in Salah et al. (2016). More focus on comparing SOA and microservices has been studied in terms of communication protocols, messaging framework, and service discovery. In all the works stated above, there has been no empirical work done in comparing both SOA and microservices styles.

To the best of our knowledge, there are currently no publications on comparison of SOA and microservices in terms of complexity and performance. Few works have been carried out in evaluating the performance of microservices and compared with monolithic applications (Gos and Zabierowski, 2020; Ueda et al., 2016; Villamizar et al., 2017). However, the focus of our work is to compare the microservices architecture with SOA style. In this work, we consider the complexity and performance testing as attributes for comparison between both the architectures.

4 Preliminaries

We define a formal model called SG which resembles any service-based application. By considering the inputs and outputs from the application programming interfaces (APIs) of the application, we create a SG. As both SOA and microservices architectures have services as the main component, we use this SG for comparison of both the architectural styles.

4.1 Service graph

SG is a standard graph created for the visual analysis of communication and dependence between the services of the application. It helps in extracting the values for metrics such as NoSs and the complexity of each service. It also helps in software engineering tasks such as effort estimation, complexity analysis, design patterns, etc. The generalised form of a SG is shown in Figure 1.

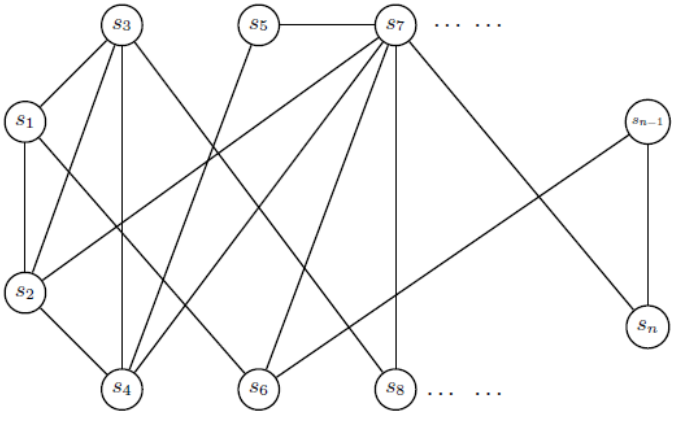
4.1.1 Service definition

Let a graph $G(V, E)$ be a SG with **n** nodes, where the nodes of the graph represent a set of services in the application, and edges between the nodes represent the interactions or dependency each service has with other services in the application. Let $V = \{s_1, s_2, s_3, ...\}$ be the nodes of the SG where s_1, s_2, s_3, ... are services and $E = \{(s_1, s_2), (s_1, s_3), (s_2, s_4), ...\}$ be the edges between the nodes which represent the dependency between the services. A service can be represented as a set of coordinating and interacting processes as defined in equation (1).

$$S_i =< P_1{}^i, P_2{}^i, P_3{}^i, ..., P_n{}^i, \Lambda > \quad (1)$$

where S_i is the logical service instance, $P_k{}^i$ indicates k^{th} process implementing logical service functionality f_i through the programmatic interface I_i and Λ represents network communication function between individual processes (Yanchuk et al., 2006).

Figure 1 Formal representation of service-based application



5 Case study application

We use a standard web-based application, vehicle management system (VMS) (Bhallamudi et al., 2009) which is used to select, customise, and purchase vehicles and its parts using a web interface. The goal of this application is to help customers to select, customise, compare vehicle, locate dealer and request for a quote. All the details of the vehicles, their parts and prices are configured in the database and helps customers with details using the user interface. Customers can select the vehicle of choice and the dealer for the selected vehicle from the inventory data. Customers can even select the part and the product type of the vehicle from the interface. The selected information is generated as a lead and sent to the dealer who helps the customer in purchasing the vehicle and its parts.

5.1 SOA-based application

The chosen case study application has eight services in the SOA implementation. The details of the SOA services are listed in Table 1 and the SG representation denoted as SG_SOA is presented as shown in Figure 2. We implement the SOA-based application using The Information Bus Company (TIBCO) Business Works (BW) and deployed using TIBCO administrator. For data storage, oracle database is considered and database palettes of TIBCO BW helps in connecting to the database. The communication between the services is using REST protocol via HTTP. Each service is deployed as an independent archive in a single server.

Figure 2 SG_SOA: SG representation of SOA-based web application

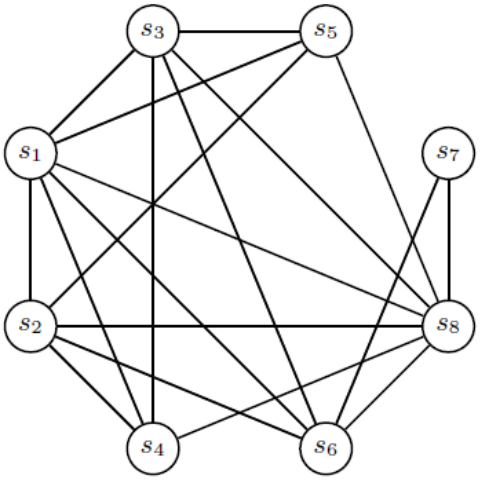


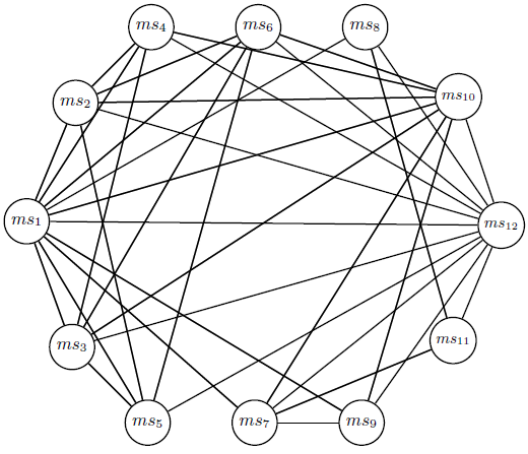
Table 1 Details of services of both SOA and microservices-based applications

Notation in SG_SOA	*SOA services*	*Microservices*	*Notation in SG_MSA*
s_1	Config service	Config service	ms_1
s_2	Part service	Part service	ms_2
s_3	Product service	Product service	ms_3
s_4	Compare service	Compare service	ms_4
s_5	Incentives and pricing service	Incentives service	ms_5
		Pricing service	ms_6
s_6	Dealer and inventory service	Dealer service	ms_7
		Dealer locator service	ms_8
		Inventory service	ms_9
s_7	Lead service	Get-a-quote service	ms_{10}
		Lead processor service	ms_{11}
s_8	User interface client	User interface client	ms_{12}

5.2 Microservices-based application

In order to develop microservices-based application, we adopt the microservices extration approach from SOA-based application (Raj and Ravichandra, 2020) and generate the SG which helps in identifying the candidate microservices. Considering the set of microservices, the chosen case study application is implemented using spring boot framework and REST/JSON formats are used for communication among the services in the network. Eureka service is used as service registry to store all the services. To store the data, MYSQL database is considered and spring boot connector retrieves data using JPA connector. Each microservice is deployed with containers using docker in the cloud. The docker image of the application is created, deployed in docker hub and containers are created from docker images. The details of the generated microservices are presented in Table 1 and the SG (SG_MSA) is presented in Figure 3.

Figure 3 SG_MSA: SG representation of microservices-based web application



6 Complexity analysis

As discussed, there is a strong need to compare both SOA and microservices architecture as software architects are in chaos whether to continue the applications in SOA or to migrate them to microservices style. We consider the metrics for measuring the complexity of the software architectures (Zhao, 1998) and extract the metric values from SG representation. The metric values are considered to compare the complexity of both the architectures. Below, the metric definitions and evaluation of both the architectures are discussed.

6.1 Total complexity of the software architecture

This metric is used to calculate the total complexity of the application by considering the dependencies between the services. Let D_t be the set of all dependencies in the SG of the software architecture, the total complexity M_T of the architecture can be measured as

$$M_T = \mid D_t \mid \tag{2}$$

The dependencies between the services are extracted from the SG representation.

6.2 Global complexity of the software architecture

Each service may contain multiple processes and each process also adds to the complexity of the system. By considering the individual complexities of the services, we consider another metric for calculating the global complexity.

Let M_T be the total complexity and $M_1, M_2, ..., M_k$ be the individual service complexities, then the global complexity M_G is calculated as

$$M_G = M_T + \sum_{i=1}^{k} M_i. \tag{3}$$

To calculate the individual complexities of each service in the application, the coupling factor of the service is considered.

6.3 Service coupling

The coupling between the services indicates the dependencies it has on other services and it should be always low. The more coupling intensity between the services, the high is the complexity of the system. Hence, we consider the metrics related to coupling to calculate the complexity of individual services.

6.3.1 Metrics related to coupling

SG provides the details of basic metrics that are used to determine other metric values. *NoSs* value is given by the count of nodes in the SG,

$$NoS = n \tag{4}$$

Coupling of services (CS) value is given by the degree of each node as given in equation (5).

$$CS_i = deg(s_i) \tag{5}$$

Relative coupling of services (RCS) denotes the degree of coupling in a particular service (Qingqing and Xinke, 2009). RCS of service is calculated using the formula in equation (6).

$$RCS[s] = \frac{CS[s]}{NoS} \tag{6}$$

The complexity of the application can be measured with this metric. The coupling intensity of a service is directly proportional to the value of the RCS. Using the above metrics, we evaluate the chosen case study application and compare both the architectural styles.

6.4 SOA-based application

In order to compare both the styles, we extract the metric values from the SG of both the styles. From the SG representation of SOA style as shown in Figure 2, we identify the dependencies and also calculate the CS and RCS values, presented in Table 2.

6.4.1 Total complexity

The total complexity of the application is calculated by the summation of all dependencies among the services. By considering the CS values from Table 2, we calculate the total complexity.

$$M_T = \mid D_t \mid = 38$$

Table 2 List of services with CS and RCS values of SOA-based application

Service #	*Interacting services*	*CS value*	*RCS value*
s_1	2, 3, 4, 5, 6, 8	6	0.75
s_2	1, 4, 5, 6, 8	5	0.62
s_3	1, 4, 5, 6, 8	5	0.62
s_4	1, 2, 3, 8	4	0.5
s_5	1, 2, 3, 8	4	0.5
s_6	1, 2, 3, 7, 8	5	0.62
s_7	6, 8	2	0.25
s_8	1, 2, 3, 4, 5, 6, 7	7	0.87

6.4.2 Global complexity

The global complexity metrics require the complexities of individual services and hence, we consider the RCS values for measuring the individual service complexity.

$$M_G = M_T + \sum_{i=1}^{8} M_i. = 38 + 4.73 = 42.73$$

6.5 Microservices-based application

Similarly, from the SG of microservices application, we extract the dependencies from the SG as shown in Figure 3 and calculate the CS and RCS values, presented in Table 3.

6.5.1 Total complexity

The total complexity of the application is calculated by the summation of all dependencies in the SG of microservices-based application. By considering the CS values from Table 3,

$$M_T = \mid D_t \mid = 70$$

6.5.2 Global complexity

The global complexity metrics require the complexities of individual services and hence, we consider the RCS values for measuring the individual service complexity. From the RCS values in Table 3,

$$M_G = M_T + \sum_{i=1}^{12} M_i. = 70 + 5.8 = 75.8$$

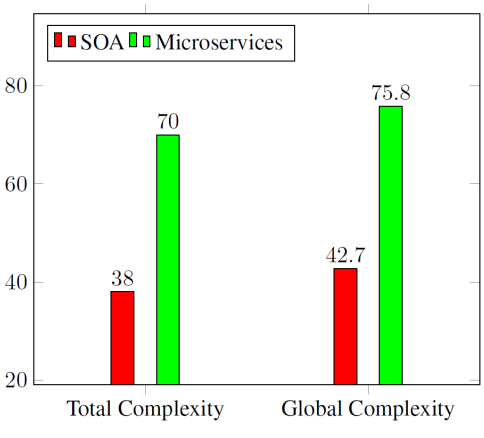
Table 3 List of services with CS and RCS values of microservices-based application

Service #	*Interacting services*	*CS value*	*RCS value*
ms_1	2, 3, 4, 5, 6, 7, 9, 10, 12	9	0.75
ms_2	1, 4, 5, 6, 10, 12	6	0.5
ms_3	1, 4, 5, 6, 10, 12	6	0.5
ms_4	1, 2, 3, 10, 12	5	0.41
ms_5	1, 2, 3, 6, 12	5	0.41
ms_6	1, 2, 3, 5, 10, 12	6	0.5
ms_7	1, 9, 10, 11, 12	5	0.41
ms_8	11, 12	2	0.16
ms_9	1, 7, 10, 12	4	0.33
ms_{10}	1, 2, 3, 4, 6, 7, 9, 12	8	0.67
ms_{11}	7, 8, 12	3	0.25
ms_{12}	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	11	0.91

6.6 Comparison

The values of both total and global complexities for both SOA and microservices-based applications are plotted as a graph as shown in Figure 4. The results show that the microservices-based application is more complex when compared with SOA-based application. By the definition of microservices, the NoSs will be more in microservices than in SOA. For the chosen case study application, the NoSs in SOA application is 8 and in the microservices-based application, it is 12 whereas the sum of total dependencies is 38 in SOA application and 70 in microservices-based application. We observe that 50% increase in NoSs has lead to 84% increase in the dependencies in the microservices application. The more NoSs, the more will the complexity of the application. From the SGs as shown in Figures 2 and 3, we can observe that the dependencies among the services in microservices application is very high compared to SOA-based application.

Figure 4 Comparison of complexities (see online version for colours)



7 Performance testing

In order to compare the performance of both the architectures, we consider load testing on both the applications. As JMeter can be used to carry out performance tests for SOAP and REST-based web services, we configured JMeter to perform the load testing by considering 500 and 1,000 users of load on both the applications. All the services in both SOA and microservices-based applications are given load using JMeter tool and we capture the average response time for 500 and 1,000 users separately for all services. The time taken from sending the request to getting the first response is treated as response time and it is measures in millisecs. The average response time of both SOA-based services and microservices are presented in Tables 4 and 5 respectively.

Table 4 Response time values of SOA services

SOA services	*Response time (millisecs)*	
	500 users	*1,000 users*
s_1	3,412.76	6,850.68
s_2	4,519.34	8,080.58
s_3	6,127.12	10,870.84
s_4	5,923.67	10,381.83
s_5	7,534.18	15,127.50
s_6	8,316.41	16,199.60
s_7	5,681.75	13,887.48
s_8	6,120.67	11,812.99

Table 5 Response time values of microservices

Microservices	*Response time (millisecs)*	
	500 users	*1,000 users*
ms_1	2,691.47	7,353.89
ms_2	4,245.85	14,590.98
ms_3	5,381.52	6,712.26
ms_4	4,818.51	14,296.40
ms_5	3,133.11	3,481.54
ms_6	4,914.38	9,608.29
ms_7	4,608.70	5,398.62
ms_8	4,037.72	5,218.80
ms_9	4,403.81	3,783.48
ms_{10}	2,308.32	4,839.70
ms_{11}	3,671.6	10,303.73
ms_{12}	4,866.44	9,376.96

7.1 Comparison criteria

In order to compare the performance of both the architectural styles, we define BRs according to the functionality of the case study application. The sequence of services invoked for processing a particular business functionality are stored as a sequence with the service numbers. By understanding the complete case study application, we identfied seven major BRs and listed them in Table 6 for SOA and microservices-based applications respectively. The time taken for each of these BRs in both SOA and microservices-based applications are considered as criteria for comparison.

Table 6 Mapping of business requests with workflows

Business requests	*Sequence of SOA services*	*Sequence of microservices*
BR1	$s_8 - s_1 - s_3 - s_2 - s_5 - s_6$	$ms_{12} - ms_1 - ms_3 - ms_2 - ms_6 - ms_5 - ms_9$
BR2	$s_8 - s_6 - s_3 - s_2 - s_4$	$ms_{12} - ms_9 - ms_3 - ms_2 - ms_4$
BR3	$s_8 - s_6 - s_3 - s_2 - s_5 - s_4$	$ms_{12} - ms_9 - ms_3 - ms_2 - ms_6 - ms_5 - ms_4$
BR4	$s_8 - s_6$	$ms_{12} - ms_8 - ms_9$
BR5	$s_8 - s_6 - s_7$	$ms_{12} - ms_9 - ms_{10} - ms_{11} - ms_8 - ms_7$
BR6	$s_8 - s_1 - s_6$	$ms_{12} - ms_1 - ms_7 - ms_8$
BR7	$s_8 - s_1 - s_5$	$ms_{12} - ms_1 - ms_5 - ms_6$

To better understand the BRs, lets consider the business functionality of comparing two different vehicle models and their parts. The comparing of products and parts of the vehicles is configured as BR BR2. The user through the web interface client selects the product and parts of the vehicles from inventory and chooses the compare option in the application. The results of the comparison are displayed on the screen and the user decides on the vehicle to select. The services which gets invoked in this process are represented as a sequence, shown in Table 6.

7.2 Comparison results

The average response times of each of the services involved in the BRs are added to get the response time of the complete BR. The time taken for all the requests under 500 and 1,000 users for both SOA and microservices applications are plotted as a tornado graph as shown in Figures 5 and 6. It is clear from the graph that the average response time for completing the BRs in SOA-based application is high compared to microservices application. For in detail analysis of the performance load testing, we consider two different scenarios w.r.t. to the NoSs involved in getting the response of a BR.

7.2.1 BR having the same NoSs

From the details of BRs and corresponding sequences in Table 6, we can observe that BR2 has the same NoSs and the functionality of the service also remains the same but designed with different architectural styles. We consider this scenario to test the response time for BR2 which clearly presents the impact of architecture and its environmental

factors. The BR BR2 is verified using 500 and 1,000 users and the average response times are plotted as a bar graph as shown in Figure 7.

Figure 5 Response time of BRs for 500 users (see online version for colours)

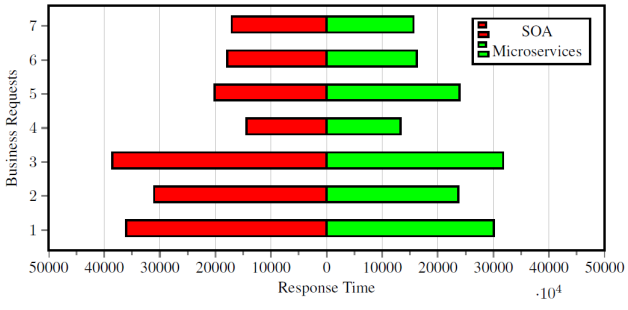


Figure 6 Response time of BRs for 1,000 users (see online version for colours)

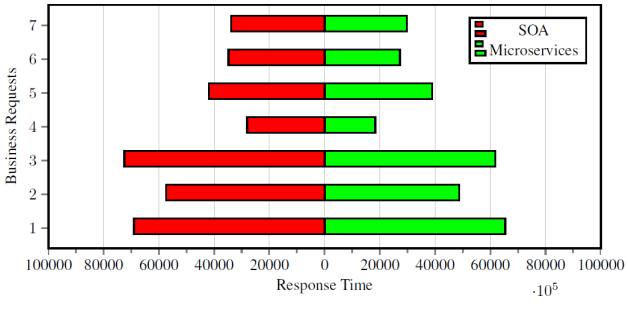
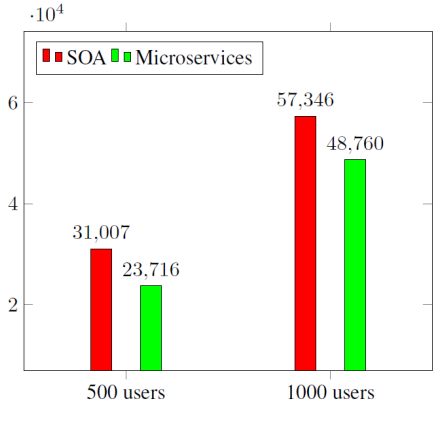


Figure 7 Average response time for BR2 (see online version for colours)

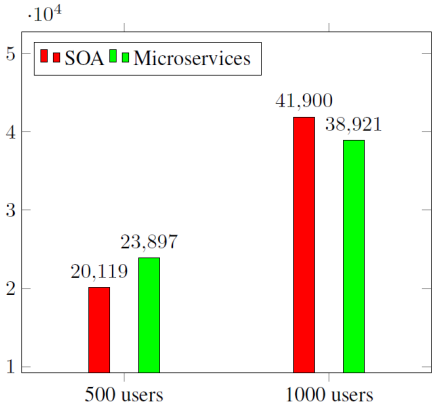


The results from the graph for the BR BR2 of our case study show that the microservices-based application has better reponse time when compared to SOA-based application even though the NoSs and the business functionality remains same.

7.2.2 BR having different NoSs

To identify how both the architectures behave with different NoSs, we choose the BR BR5 as NoSs involved in SOA-based application is three and in microservices-based application is six. The BR BR5 is also tested with 500 and 1,000 users for both the applications and results are plotted as a bar graph as shown in Figure 8. The results show that microservices have high response time when compared to SOA with 500 users and the response time is better for microservices when the number of users is increased to 1,000. It shows that though the NoSs is high in microservices, because of its cloud-based deployment environments, the average response time of microservices-based application is low.

Figure 8 Average response time for BR5 (see online version for colours)



8 Conclusions

After observing the advantages of migrating monolithic applications to microservices, migration of SOA-based applications has gained an interest in the IT industry. However, some of the software architects are in chaos whether to migrate the SOA-based applications or not, as they are unclear about the pros and cons of migration to microservices. Moreover, there is no much empirical work done in comparison of both these architectural styles. This work helps in understanding the differences in terms of complexity and performance of both the styles. Based on the analysis done for the complexity of the applications, it is clear that microservices application has more NoSs and application is more complex than SOA. The coupling between the services is also less in microservices architecture. However, the performance testing shows better results for microservices with quick response times with 500 and 1,000 users. Furthermore, the chosen case study application exhibits better results when chosen the BRs with the same NoSs in both styles. After this experimental study, we conclude that microservices architecture exhibits better performance results with the use of cloud-based environments and can be used in the design of large enterprise applications.

In this work, we compared both the architectures in terms of coupling, complexity, and performance. As part of future work, other features such as scalability, maintenance, etc. will be considered for comparison.

References

AlSharif, M., Bond, W.P. and Al-Otaiby, T. (2004) 'Assessing the complexity of software architecture', in *Proceedings of the 42nd Annual Southeast Regional Conference*, 2 April, pp.98–103.

Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M. and Steinder, M. (2015) 'Performance evaluation of microservices architectures using containers', in *2015 IEEE 14th International Symposium on Network Computing and Applications*, IEEE, 28 September, pp.27–34.

Balalaie, A., Heydarnoori, A. and Jamshidi, P. (2016) 'Microservices architecture enables devops: migration to a cloud-native architecture', *IEEE Software*, 18 March, Vol. 33, No. 3, pp.42–52.

Bhallamudi, P., Tilley, S. and Sinha, A. (2009) 'Migrating a web-based application to a service-based system-an experience report', in *2009 11th IEEE International Symposium on Web Systems Evolution*, IEEE, 25 September, pp.71–74.

Bogner, J., Wagner, S. and Zimmermann, A. (2017) 'Automatically measuring the maintainability of service-and microservice-based systems: a literature review', in *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, 25 October, pp.107–115.

Bogner, J., Wagner, S. and Zimmermann, A. (2019) 'Using architectural modifiability tactics to examine evolution qualities of service-and microservice-based systems', *SICS Software-Intensive Cyber-Physical Systems*, 1 June, Vol. 34, Nos. 2–3, pp.141–149.

Bogner, J., Zimmermann, A. and Wagner, S. (2018) 'Analyzing the relevance of SOA patterns for microservice-based systems', *Zeus*, February, Vol. 9, pp.9–16.

Cerny, T., Donahoo, M.J. and Pechanec, J. (2017) 'Disambiguation and comparison of SOA, microservices and self-contained systems', in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, 20 September, pp.228–235.

Cerny, T., Donahoo, M.J. and Trnka, M. (2018) 'Contextual understanding of microservice architecture: current and future directions', *ACM SIGAPP Applied Computing Review.*, 29 January, Vol. 17, No. 4, pp.29–45.

Di Francesco, P., Malavolta, I. and Lago, P. (2017) 'Research on architecting microservices: trends, focus, and potential for industrial adoption', in *2017 IEEE International Conference on Software Architecture (ICSA)*, IEEE, 3 April, pp.21–30.

Escobar, D., Cárdenas, D., Amarillo, R., Castro, E., Garcés, K., Parra, C. and Casallas, R. (2016) 'Towards the understanding and evolution of monolithic applications as microservices', in *2016 XLII Latin American Computing Conference (CLEI)*, IEEE, 10 October, pp.1–11.

Fowler, M. and Lewis, J. (2014) *Microservices A Definition of this New Architectural Term* [online] http://martinfowler.com/articles/microservices.html (accessed 22 March 2014).

Gos, K. and Zabierowski, W. (2020) 'The comparison of microservice and monolithic architecture', in *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, IEEE, 22 April, pp.150–153.

Gribaudo, M., Iacono, M. and Manini, D. (2017) 'Performance evaluation of massively distributed microservices based applications', in *31st European Conference on Modelling and Simulation, ECMS 2017*, European Council for Modelling and Simulation, pp.598–604.

Jamshidi, P., Pahl, C., Mendonça, N.C., Lewis, J. and Tilkov, S. (2018) 'Microservices: the journey so far and challenges ahead', *IEEE Software*, 4 May, Vol. 35, No. 3, pp.24–35.

Krause, L. (2015) *Microservices: Patterns and Applications: Designing Fine-Grained Services by Applying Patterns § hLucas Krause*, Lucas Krause.

Larrucea, X., Santamaria, I., Palacios, R. and Ebert, C. (2018) 'Microservices', *IEEE Software*, Vol. 35, No. 3, pp.96–100.

Lassing, N., Bengtsson, P., Van Vliet, H. and Bosch, J. (2002) 'Experiences with ALMA: architecture-level modifiability analysis', *Journal of Systems and Software*, 1 March, Vol. 61, No. 1, pp.47–57.

Nevedrov, D. (2006) *Using JMeter to Performance Test Web Services*, 1 August, dev2dev.

Pahl, C., Jamshidi, P. and Zimmermann, O. (2018) 'Architectural principles for cloud software', *ACM Transactions on Internet Technology (TOIT)*, 2 February, Vol. 18, No. 2, pp.1–23.

Qingqing, Z. and Xinke, L. (2009) 'Complexity metrics for service-oriented systems', in *2009 Second International Symposium on Knowledge Acquisition and Modeling*, IEEE, 30 November, Vol. 3, pp.375–378.

Rademacher, F., Sachweh, S. and Zündorf, A. (2018) *Analysis of Service-Oriented Modeling Approaches for Viewpoint-Specific Model-Driven Development of Microservice Architecture*, 26 April, arXiv preprint arXiv:1804.09946.

Raj, V. and Ravichandra, S. (2018) 'Microservices: a perfect SOA based solution for enterprise applications compared to web services', in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, IEEE, 18 May, pp.1531–1536.

Raj, V. and Ravichandra, S. (2010) 'A service graph based extraction of microservices from SOA', *Software: Practice and Experience*.

Salah, T., Zemerly, M.J., Yeun, C.Y., Al-Qutayri, M. and Al-Hammadi, Y. (2016) 'The evolution of distributed systems towards microservices architecture', in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, IEEE, 5 December, pp.318–325.

Salah, T., Zemerly, M.J., Yeun, C.Y., Al-Qutayri, M. and Al-Hammadi, Y. (2017) 'Performance comparison between container-based and VM-based services', in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, IEEE, 7 March, pp.185–190.

Spitznagel, B. and Garlan, D. (1998) 'Architecture-based performance analysis', in *Proceedings of the 1998 Conference on Software Engineering and Knowledge Engineering (SEKE'98)*.

Taibi, D., Lenarduzzi, V. and Pahl, C. (2017) 'Processes, motivations, and issues for migrating to microservices architectures: an empirical investigation', *IEEE Cloud Computing.*, September, Vol. 4, No. 5, pp.22–32.

Taibi, D., Lenarduzzi, V. and Pahl, C. (2018) 'Architectural patterns for microservices: a systematic mapping study', in *8th International Conference on Cloud Computing and Services Science, CLOSER*.

Ueda, T., Nakaike, T. and Ohara, M. (2016) 'Workload characterization for microservices', in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, 25 September, pp.1–10.

Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., Zambrano, A. and Lang, M. (2017) 'Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures', *Service Oriented Computing and Applications*, June, Vol. 11, No. 2, pp.233–247.

Vural, H., Koyuncu, M. and Guney, S. (2017) 'A systematic literature review on microservices', in *International Conference on Computational Science and Its Applications*, Springer, Cham, 3 July, pp.203–217.

Wang, S., Ding, Z. and Jiang, C. (2020) 'Elastic scheduling for microservice applications in clouds', *IEEE Transactions on Parallel and Distributed Systems*, 27 July, Vol. 32, No. 1, pp.98–115.

Wilde, N., Gonen, B., El-Sheikh, E. and Zimmermann, A. (2016) 'Approaches to the evolution of SOA systems', in *Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures*, pp.5–21, Springer, Cham.

Wolff, E. (2016) *Microservices: Flexible Software Architectures*, CreateSpace Independent Publishing Platform, USA.

Xiao, Z., Wijegunaratne, I. and Qiang, X. (2016) 'Reflections on SOA and microservices', in *2016 4th International Conference on Enterprise Systems (ES)*, IEEE, 2 November, pp.60–67.

Yanchuk, A., Ivanyukovich, A. and Marchese, M. (2006) 'Towards a mathematical foundation for service-oriented applications design', *Journal of Software*, July, Vol. 1, No. 1, pp.32–39.

Zhao, J. (1998) 'On assessing the complexity of software architectures', in *Proceedings of the Third International Workshop on Software Architecture*, 1 November, pp.163–166.

Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Li, W. and Ding, D. (2018) 'Fault analysis and debugging of microservice systems: industrial survey, benchmark system, and empirical study', *IEEE Transactions on Software Engineering*, 18 December.