The Evolution of Distributed Systems Towards Microservices Architecture

Tasneem Salah, M. Jamal Zemerly, Chan Yeob Yeun, Mahmoud Al-Qutayri, Yousof Al-Hammadi
Department of Electrical and Computer Engineering
Khalifa University of Science Technology and Research
PO Box 127788 Abu Dhabi, UAE
{tasneem.salah, jamal, cyeun, mqutayri, yousof.alhammadi}@kustar.ac.ae

Abstract— **Applications developed to fulfil distributed systems needs have been growing rapidly. Major evolutions have happened beginning with basic architecture relying on initiated request by a client to a processing side referred to as the server. Such architectures were not enough to cope up with the fast ever-increasing number of requests and need to utilize network bandwidth. Mobile agents attempted to overcome such drawbacks but did cope up for so long with the growing technology platforms. Service Oriented Architecture (SOA) then evolved to be one of the most successful representations of the client-server architecture with an added business value that provides reusable and loosely coupled services. SOA did not meet customers and business expectations as it was still relying on monolithic systems. Resilience, scalability, fast software delivery and the use of fewer resources are highly desirable features. Microservices architecture came to fulfil those expectations of system development, yet it comes with many challenges. This paper illustrates how distributed systems evolved from the traditional client-server model to the recently proposed microservices architecture. All architectures are reviewed containing brief definitions, some related work and reasoning of why they had to evolve. A feature comparison of all architectures is also provided.**

Keywords- Distributed Systems; Microservices; Mobile Agents; Service Oriented Architecture

I. INTRODUCTION

Most of the IT applications seen today are distributed due to the evolution of highly available communication networks [1]. Those applications vary from local networks to large-scale networks such as those relying on the Internet. Applications nowadays are extended over a wide range of physical locations or nodes including organizations. The development of Distributed Software Systems began to increase rapidly in the last decade due to the need to deploy services to a wide range of people online and in real time [2]. The development process was centralized over the quality attributes of the software, which include how much the service is reusable, reliable, and always available and meets end-user requirements [3]. Relying on a common communication interface across nodes became essential due to the growth of heterogeneous environments. It is important to start with the traditional architectures before discussing what is most trendy and recommended to use nowadays. As seen in Fig.1, paradigms have evolved for structuring applications based on the demands of their current period of time. Originally, systems started without considering networking layer to allow other devices to communicate with each other. With the growth of networking capabilities, the client-server model rose to consider various users to communicate and demand for services from servers.

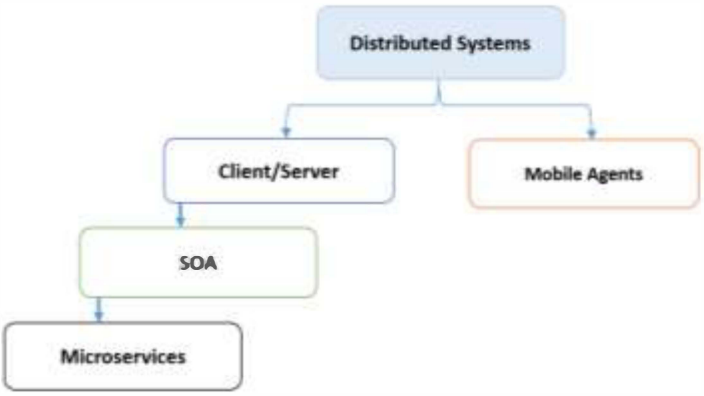

Figure 1. Distributed Systems taxonomy evolution representing architectures relied on in this survey

All the processing was meant to be at the server side in the past due to the lack of end-user resources and features for computing abilities. In the past decade, client-server model became one of the most popular distributed systems paradigms which relied on Remote Procedure Calls (RPC) as a communication method. Many paradigms then evolved aiming to handle modern applications as they were enhancing rapidly over the years. Mobile Agents technology is one of the paradigms that were proposed to overcome slow networks as computation was meant to be moved to the server side. Mobile agents were developed to handle complex systems that require simultaneous connection with the source of information (servers) in order to keep track of updates. This paradigm got less attention with the growth of efficient communication links and the demand to have business software structures. Client-server again gained more attention to be developed as its major concern of high network bandwidth consumption declined with the growth of available and fast network connections and lightweight communication protocols (APIs).

Service Oriented Architecture (SOA) was proposed to provide integration solutions for distributed services across different

978-1-908320/73/5/$31.00 © 2016 IEEE

organizations and operational systems. SOA relies on a simple type of Remote Procedure Calls such as SOAP (Simple Object Access Protocol) which is often used. This architecture also aimed at utilizing services across the web. The client basically communicates a message with a middleware using a common communication language and it gets translated to reach to the service communicating with different message types. More lightweight communication methods were then needed to satisfy business models demands including scalability, reliability and efficient use of resources. Such paradigms were proposed aiming to rely only on API calls among services due to their lightweight and simplicity. The values of API's and the reasons why businesses are relying on them are discussed in [4]. One of the main reasons is agility, which results in their lightweight communication. This generally explains the triggered growing interest in Microservices architecture. We focus in this paper on how systems began from basic Client-Server architecture to evolving towards Microservices. It is important to address why there was a need to evolve toward such architecture and to whom it can be beneficial.

The remainder of the paper is organized as follows: Section II describes the Client-Server Paradigm. Section III discusses Mobile Agents. Section IV explains the Service Oriented Architecture. Then the Mircoservices Architecture is discussed in Section V. For each of the architectures an introduction, related work and drawbacks are provided. A comparison of features of all discussed architectures is provided in Section VI. Finally, Section VII concludes the review.

II. CLIENT-SERVER PARADIGM

Client-Server model is one of the most popular paradigms relied on in distributed systems [5]. It is a way to represent the relationship between two programs communicating with each other, one is called the client and the other is the server. Usually, the client is the one initiating the communication by a request and the server is the one receiving it. Both the client and the server can be located on the same machine or each on different machines. The server is often the one having the resources represented as databases or applications to be shared with the requester (client) [6]. The request-response type of protocol of this model is referred to as remote procedure calls (RPC). Client-Server paradigm was mainly used in Java environments relying on Remote Object invocation mechanism [5]. The technology had a huge impact on the Client-Server model and made it evolve from two tier architecture to three and n-tier architectures. Figure 2 provides a rough explanation of the basic structure of the client-server model.

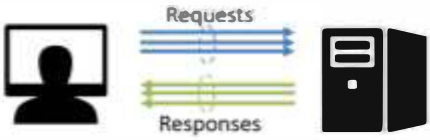

Figure 2. Basic parts and functions of Client-Server Paradigm

A. Related Work

One of the major concerns of the client-server model was the limited bandwidth allowed for clients to use over the network. Many efforts came in order to solve this matter using different techniques. The authors in [8] proposed a model to maintain efficient bandwidth utilization by reducing the amount of calls between the server and the client and amount of sent data. The model focuses on the use of 'eBUCS' protocol as it creates Virtual Client Database. This technique collaborated in creating the database on the client's side rather than reaching the server side for each request. The design focuses on adding intelligence part to the protocol in order to overcome the probability of frequent updates to occur to the database.

Peer-to-peer paradigm then came in order to ease file-sharing among different hosts other than the server itself to reduce latency caused of the centralized server [9]. This paradigm took a huge role in many research areas [10-11]. However, despite the massive advantages and high speed this architecture offers, it cannot be used as a reliable communication method, due to the major security flaws including the interaction with untrusted hosts [12]. Huge delay in transmission can occur even in peer-to-peer paradigm especially when the number of nodes increases causing a saturated network. Hybrid architectures involving peer-to-peer along with several servers came to overcome the limitations of client-server and peer-to-peer in the context of performing higher processing [9]. Cloud based architectures then came to cause a tremendous impact on the computing era. These architectures offer management support for network resources to maintain scalability and consistency [9].

B. Drawbacks

Although, client-server model can be flexible, robust, and scalable [13], it yet suffers from many issues [7]:

- Scalability takes time: To scale the service available on the server side, a whole server machine needs to be added in order for the service to scale.
- The way this model scales will raise another disadvantage, which is the hardware cost.
- Frequent updates are hard to maintain: Mostly maintenance is done by the administrator side and it is difficult to distribute the maintenance duty among different personnel.
- Procedure calls done by the clients can affect the load on the network.
- The inability to integrate autonomous services as a whole to be requested by various clients (Supported by SOA architecture - see Section IV).
- Services can be hijacked by enormous number of requests leading to the service to be unavailable.

Before nearly a decade, many evolutions of client-server model were proposed. Some aimed to overcome unreliable communication links by sending an agent to the source of information and reduce bandwidth consumption as proposed in [14]. Others recommended the support of stand-alone applications [13] by adding a middleware between the servers and various clients enabling them to integrate with the system. This approach is followed in Service Oriented Architecture which will be explained in Section IV.

III. MOBILE AGENTS

Mobile agents are one of the recent paradigms that came after client-server model in order to propose an improved structuring mechanism for distributed applications. Basically, an agent is a piece of software having the ability to move with its code, data and state from one host to another to fulfil a

specific task. As in [15], mobile agents move freely between hosts (which have to be secure), travel publicly in an encrypted form. They consist of three parties: Mobile agent's owner, locations visited by mobile agents and the adversary as illustrated in Figure 3. An agent is composed of three parts which also take place in travelling: Data, State and Code.

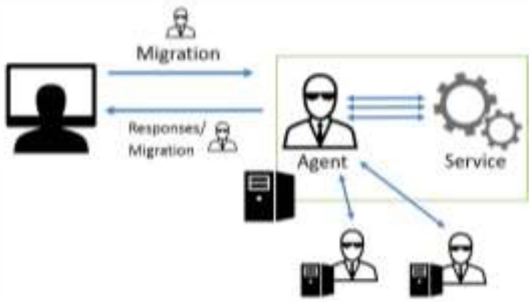

Figure 3. Basic parts and features of Mobile Agents architecture

Authors in [5] stated that the data part evolves based on the visited hosts containing information to be collected. The state of an agent is also saved in order to allow it continue its execution on the destination host to be moved to. Usually, mobile agents execute from the beginning on each host resulting in withdrawing the consideration of the previous execution state. Finally, the code part is where the algorithm of the agent resides.

A. Related Work

One of the major goals of using mobile agents is to overcome highly dynamic environments by moving the computation part to the source of information. It is beneficial in a way the agents can hold the type of algorithm needed to be applied on the data to return back what is only needed for the client. This can be seen as illustrated by the authors in [15] as this architecture was used in stock market predictions because of the difficulty for humans to control global market interactions and handle information gathering in such complex environments. Mobile agents can also overwhelm slow networks as they save communication costs by reducing the amount of queries between the requester and the receiver. The entire computation is moved to the host consisting of the target data. In addition, mobile agents are good at decision making [15]. They are on high demand especially for complex environments such as stock markets and recommender systems. The authors in [15] stated that placing those mobile agents in critical applications will increase the attraction towards attacking them. The paper proposes solutions in order to avoid attacks resulted from traffic analysis.

A minimal mobile agent model was developed in [5] depending on a popular system for mobile agents called Aglets mobile agents platform. The platform was simple and contained the basic functions for the purpose of comparing the prototype with the client-server model. An investigation was done on how to achieve web service integration in a pervasive network using mobile agents [7]. The aim is to make web services fast to get especially when huge number of clients is connected and requesting those services. MAs were found to be useful and take less communication load especially when exchanging multimedia data. The execution time for both RPC and MA is computed and compared and resulted in MA requiring less time. The concept of mobile cloud computing was also proposed in [16] to mainly overcome power and battery life limitations of mobile devices. Mobile agents can also contribute to offer security features to some systems. A security scheme is represented relying on mobile agents due to their flexibility and lightweight software [17]. Multiple agent types were proposed to be triggered locally once the cloud service provider response to a user request including intrusion detection agents, resource monitoring agent and others. Mobility of agents was required for evidence collecting of vulnerable and suspicious virtual machines. Many other applications hosted the mobile agent technology.

B. Drawbacks

Mobile agent paradigm was proposed as a viable solution to overcome client-server paradigm limitations as seen in the related work (Part A). Unfortunately, they did not meet the level of expectations they were made for about a decade ago due to the following reasons:

- They were intended to be used over slow networks [5], and often getting offline. Nowadays, facing such problems is avoidable relying on advanced networking methods.
- They were made for client customization purposes; it became possible to add such features in remote calls.
- It requires to take into account major security measures as trusted host, trusted MA community, secure communication protocols and secure MA (to not expose owner profile).
- Not being able to scale; acceptance of executable code hosting and the need for common execution language.
- Hosts should have mobile agent hosting capability which is not widely spread among many machines.
- Agent classes need to be installed on the machines [5]
- Available bandwidth is no longer an issue compared to the need of fast response and execution time
- In [18], Agents are known for their intelligence and autonomous behavior. They can contribute in decision making such as in stock market and trading decisions, but having this agent mobile, is not a necessity anymore.

IV. SERVICE ORIENTED ARCHITECTURE

Service Oriented Architecture (SOA) is another evolution of Client Server model and said to be the most successful. SOA allows services to be loosely coupled, reusable and dynamically assembled which supports the changing business environment [5]. SOA supports explicit boundaries between autonomous services located on different servers to fulfil application requirements. SOA came in order to overcome the challenges of having a large monolithic applications and aims at enhancing the reusability of a service by multiple end-user application [19]. Figure 4 illustrates the basic structure of SOA in an example.

Referring to Figure 4, a typical use case would be when clients start requesting a service through any user interface method. Then the request will be handled through an Enterprise Service Bus (ESB) as it aims on making the physical network transparent to the application by unifying services directly with the end-user [20]. ESB includes a service registration mechanism to link together widely located services. It takes the request of the end-user to be translated to the suitable message type to contact each service. This offers services from different organizations to maintain reusability of their services for various numbers of clients and customers.

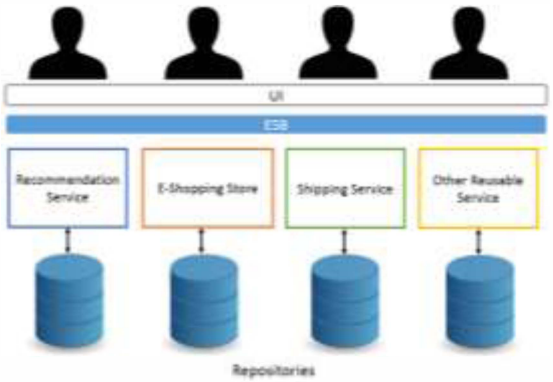

Figure 4. Illustrative example of SOA showing layers and parts relied on

A. Related Work

SOA is usually referred to as a business concept enforcing standards among different ownership domains to utilize distributed capabilities [21]. SOA was compared to three alternative architectures [21]: Application Integration, Workflow Integration and Desktop Integration. SOA was concluded to offer improved efficiency and changeability features. SOA can also be used for the purpose of wrapping up loosely coupled web services [22]. Services should share a common standard in order to conduct interaction. In [22], the various SOA protocol stacks are represented in different six layers: Transport, Message, Description, Management, Assemble and Presentation layers. SOA is usually relied on in huge enterprise applications including social media, e-shopping systems, e-banking, and many other online services dealing with a high number of continuous requests. For example, the authors in [23] investigated some benefits obtained when relying on SOA architecture in E-banking systems with a supporting case study done on a European Bank. Others used mobile agents in this architecture to utilize bandwidth in integrated web services environments. The authors in [3] used both, MA to reduce bandwidth and SOAP RPC to integrate web services of different companies to come up with unlimited satisfying services for travelers. They discuss unstable networks and a traveler application called Info Mirror which assumed that all agent platforms are the same to all information providers. They are assuming that the information provider on the internet encapsulates its services as SOAP services. In addition, programs in different languages such as JavaScript were found to be easily modified as SOAP service and it accommodates almost every platform just like HTTP. In short, they implemented a mobile agent which does SOAP-RPC calls to access information provider full services. SOAP RPC call is done within the service provider itself and not among services. The intelligent part in SOA architecture is often added to the middleware layer [19]. They are used in Event-Based applications to notify the user whenever an event occurs. These types of applications are referred to as Service Brokers. An example of service broker architecture with adoption of SOA is proposed in [24].

B. Drawbacks

The monolith representation of the reusable services seen in the SOA architecture was found not being able to keep pace of the customers and business expectations [19]. In this modern time, developing software with more features in less time and developing scalable solutions with fewer resources are highly demanded. Authors in [25] claimed that although monolithic architectures can contain several services internally, but they have to be deployed as one unit. Several copies of the same application can be run in order to maintain scalability, but they will be identical. This type of architecture is efficient to use for large applications which will make it easy to develop and deploy. The main issue that will rise of such applications is the difficulty for modifying and understanding it. Big applications contain large code base, adding updates or replacing developers for that specific task will be exhaustive and nearly impossible for some applications. Redeploying the whole application might be required for even small components to be added. The solution of running multiple copies for scaling is not efficient as it will result in the increase of memory consumption. In addition, the reusability of common data between the copies as caching solutions will not work. SOA typically centralizes the sophisticated operations including business constraints, message routing, and service orchestration all in the ESB [26]. Modern efforts call for the need of dumb pipes (ESB) and smart endpoints [26]. Due to the illustrated drawbacks of SOA architecture, Micro-services architecture has emerged and grabbed a lot of researchers' attention recently. Microservices architecture is discussed in details in Section V.

V. MICROSERVICES

Users nowadays expect more interactive enrichment along with dynamic user experience on a wide range of devices used by clients. Developer and organizations want to have the ability for frequent updates on their services. Accordingly, relying on monolithic applications is no longer adequate [27]. An alternative architecture proposed by researchers to fulfil modern needs is known as Microservices Architecture [28]. The aim of this architecture is to break the application into a set of smaller independent services [29]. This architecture is often compared with the SOA architecture we came across in Section IV. The most serious drawback that called the need for enhancing SOA is its centralized integration [30]. Some prefer referring to microservices as an approach to build SOA. Others stated that SOA architecture can be referred to as the monolithic representation of an application or as a compressed view of microservices. Figure 5 gives an example of an approach taken to represent microservices architecture relying on the system represented using SOA in Figure 4.

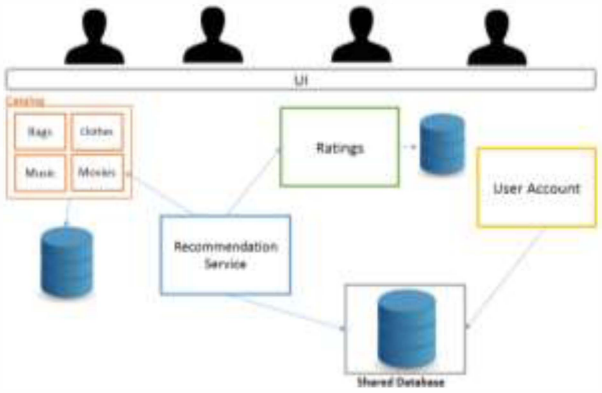

Figure 5. Illustrative example describing the evolved SOA architecture parts to microservices

In comparison with the SOA architecture model main differences with Microservices Architecture: First, the elimination of the service bus. Microservices architecture aims on maintaining dumb pipes to communicate with and moving the intelligence and control to the endpoints. This strategy enhances decoupling property between services. In addition each service is either independent or broken into smaller independent services. The figure relied on breaking the big services illustrated in Figure 4 of Section IV, into smaller independent components. It basically describes an online shopping system containing *catalog*, *recommendation*, *ratings* and *user account* independent loosely coupled services. Microservices can contain at least one functional operation to perform. The *catalog service* for example might contain other functional parts including catalogs of *bags*, *cloths*, *music* or even *movies.* Thus, each of these small services can be designed to be independent and function by their own. This eases the reusability of services as for example, the *recommender service* might only need to interact and use the *Movies catalog*. The database can also be broken into smaller databases, each containing what is beneficial to certain type of customers. This decreases the load on one database to be distributed among the smaller ones. Shared Databases among different services can be designed to be a service by its own. Since microservices will be small, they will be too many and will require the use of a lightweight communication protocol (which will be explained in part V.B).

A. Why Splitting a Monolith

Most of the related work around microservices has chosen to migrate to this architecture for the purpose of obtaining elastic scaling. Figure 6 demonstrates the major reasons behind evolving to microservices based on a number of selected reviewed papers. Most of researchers have chosen microservices to obtain scalability [26-27, 31 - 46] and other features follows to be: Fast response [32,38,44,47-48], fast software delivery [33,42,45,48-49,50,52], functional separation [25,31,34,39,51,52,54], Loose coupling [30,36,54], reusability [31,35,40-41], resilience [32-37,39], reduce resources cost [52,55-56].

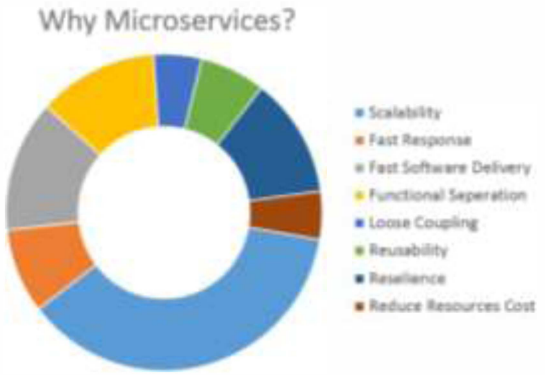

Figure 6. Percentage of features needed for systems to evolve to Microservices

Most of the publications were published between the years 2014 and 2016 which impressively shows how up to date and presently active microservices area is.

B. Development Requirements

It is worth noting that a microservice needs to be built as if it is by its own, and changing one microservice should not imply the change of the rest of the microservices. This allows the design to achieve the full advantages of deploying such architecture. Newman in his book [19] explains in details how a microservice should be built containing all the essential steps for the architecture adaption. Newman [19] illustrated that microservices allow the system to be built based on different technologies. Each microservice can be based on a different performance requirement and programming language than the others. Each microservice can also be deployed and updated independently. Illustrated below are some of the requirements to consider while developing application based on microservices.

1) Storage:

Microservices reside on Virtual Machines which enhance the scaling property. The small size of the services does not require a huge machine size to rely on and they scale independently without scaling the whole application. On the other hand, Monolith applications would need a whole new machine just to scale a minor service within the application.

Basically, virtualization is the process of splitting up the physical machine into several virtual components in which each can host different software to run within. Newman 19] illustrated an example on how to build a custom VM image. Some examples on traditional virtualization might include VMWare and AWS (Amazon Web Services) virtual machines technologies. Unfortunately building them takes a long time and some images might result with a large size. Transferring those large images across the network will result in a complex activity to be performed. The overhead is caused by the Hypervisor layer as illustrated in [19], which is a computer software layer that needs to manage all the VMs and how they access the resources. These drawbacks raised the need for container technologies such as Linux Containers [57]. They provide a separate space for the processes without the need for a hypervisor to control the machines. On top of that, Docker platform contributed on building lightweight containers and handling them as well. Containers provide a very appealing feature which is packing up the service with its dependencies into a single image also referred to as code portability [40].

Also, containers located on different machines raised the need for a tool to locate those containers in order to run them. In other words, a scheduler layer is needed (Cluster Manager).

2) Cluster Manager

As each service is located inside its own container, it is essential to handle their scheduling using a scheduler. For microservices architecture the terms '*Cluster Scheduler'* and '*Cluster Manager*' are used for this purpose. Having a cluster manager will allow resources to be used efficiently, follow user placement constraints, prevent going through pending state and to make the services always available. Basically, its task is to find which host is suitable to which container and then connect them together [58]. There are many tools used as a scheduling layer for the containers. As mentioned in [19], Google released Kubernetes and CoreOS clusters open source technologies to handle the containers. Deis [59] is another tool based on Docker. The article represented in [58] explored three popular cluster managers and came up with a detailed

comparison among them which are: Docker Swarm, Apache Mesos and Mesophere Marathon and Google Kubernetes. It was concluded that Docker Swarm is the simplest scheduler among the others due to its dependency on same API as Docker engine.

3) Service Discovery

Service discovery aims on knowing what is running on a specific environment to allow each service relying on another service knows about the other. The process takes place in two parts: The service registers itself then tries to find other services after registration. Zookeeper, Consul and Eureka are widely used among developers and often used registries for dynamic environments [19].

4) Messaging Framework

Microservices are too small and require the use of light weight communication. REST APIs are the best suited messaging exchange mechanism for microservices. The nice thing about them is that they make use of HTTP features (verbs) instead of handling independently [19]. The SOAP mechanism used in SOA architecture is not efficient to be used for microservices because of its dependency on bulky libraries for generating XML requests. REST API is proposed as a suitable lightweight communication mechanism to be used for microservices [33]. REST over HTTP can use multiple formats. Some of the well-known formats used for microservices implementations: XML and JSON. JSON is selected by most developers due to its simplicity and lightweight. Table 1 illustrates the major characteristics of the REST protocol in comparison with SOAP.

Table 1- REST vs SOAP protocol

Features	**Protocol**	
	REST	***SOAP***
Messaging Services	HTTP URL - (GET,POST,PUT and DELETE)	XML - for making requests & receiving responses
Error Handling	Using HTTP status codes	Built in
Complexity	Simple; relies on HTTP	Complex; relies on XL
Speed	Fast; simple messaging exchange with less overhead and no server sessions	Average;
Data Output	JSON (JavaScript Object Notation) format	XML within SOAP envelop

REST protocol over HTTP can use multiple formats. Some of the well-known formats used for microservices implementations: XML and JSON. JSON is selected by most developers due to its simplicity and light weight. Yet, it has to be supported by hypermedia control for some applications and they are already implemented in XML.

C. Challenges

1) Difficulty of coordination across development teams: At the beginning, each individual team has to be well trained to adapt to the new tools and techniques to consider while evolving to mivroservices architecture. From the authors experience seen in [30], referred to this task to be time consuming. In addition, the flexibility that microservices architecture offers such as having each service relying on the best suited technology stack [26] can rise the complexity of coordination between different teams. Team members should also be familiar with the supporting services of microservices such as load balancing and service registry [36].

2) Avoiding Coupled Microservices: Changing the behavior of one microservice might require building the other services which will take time [19][32]. Multiple builds for multiple microservices will be needed to avoid coupling by using one build for multiple microservices. Some cluster management tools require each service to keep track of the IP address of other services. This can have an impact on other services once a change is made to a single service. This property is referred to as Cross-Configuration as explained in [40]. The authors in [36] advice the use of consumer-driven contracts instead of service versioning to solve this matter.

3) Increased network communication: Microservices are small independent units and to interact with each other they have to communicate through a network. Having a high number of units indicates that communication will increase as well. Reliable and fast network connection is needed in this architecture. The authors in [43] mentioned the challenge of considering the operational overhead when migrating to microservices architecture. In huge systems evolving to microservices such as Netflix, Traffic would be represented as billions of API calls per day. Although tools are available to monitor and manage microservices traffic, the challenge is in whether the developer has the capability and skills to well maintain those tools for the efficiency of the system or not.

4) Need to consider adding network security: Microservices are said to be more prone to security vulnerabilities [19]. Considering authentication and data sets protection in databases are important factors which most of generic applications try to fulfil. The authors in [19] proposed ways for authentication and authorization to follow in microservices and other distributed systems. The challenge rises on handling the network side and making it secure as microservices communicate with exposed APIs to the network. An example of a methodology of safe access to host resources was provided in [40].

5) The need to split out data repositories : It is challenging to divide up a single database into smaller independent units as we discussed in this section in part V.B.2. The authors in [30] also discussed the difficulty of adapting to microservices architecture when relying on unstructured data repositories and the possibility of errors occurrence. In such cases, recovery mechanisms need to be considered to avoid data corruption scenarios.

6) Cost of monitoring will increase: The authors in [44] illustrated the difficulty of tracing the flow of requests passing through gateways and microservices in the deployment stage. The challenge of managing service state in containers was studied in [40]. It was concluded that dynamic service discovery and registry are required to ease up management and monitoring of containerized services.

7) Finding the right size and number of services: Coming up with the most efficient microservice size for an application is a tricky process. Having it too big would make it behave as a monolith application. Having them too small would raise the complexity level to handle the system despite its task

simplicity. The number of microservices to divide to also affect their size and vice versa. In some cases it is easy to just simply divide up a monolith application into a number of smaller services which seem to be relevant to the developer. Such as in [44], their enterprise monolith prototype application was divided into two services. In some applications such as the RCB (rating, charging and billing) system found in [52], the divisions and the number of services to divide to can be directly known and easy to maintain if the application was initially built with well separated functionalities from the beginning.

VI. DISTRIBUTED SYSTEMS ARCHTECTURES FEATURE COMPARISON

In this section, we compare the four architectures discussed in this paper in terms of the most demanded features to be found in developed services. Table 2 shows the features in which can be found in each architecture with respect to the other architectures.

Table 2. Comparison between distributed systems architectures

Feature	Architecture			
	Client/ Server	*Mobile Agents*	*SOA*	*Microservices*
Network Utilization	No	Yes	Yes	Yes
Inter-process communication	No	Yes	No	Yes
Elastic Scaling	No	No	No	Yes
Mostly Lightweight Communication	No	No	Yes	Yes
Resilience	No	No	Yes	Yes
Fast Software Delivery	No	No	Yes	Yes
Autonomous Service integration	No	No	Yes	Yes
Portable Services	No	Yes	Yes	Yes
Reusability of Services	No	No	Yes	Yes
Service Migration	No	Yes	No	Yes
Monolithic services	Yes	Yes	Yes	No
Loose-coupling	No	No	Yes	Yes
Mostly Cloud-based	No	No	No	Yes
Cross-functional development	No	No	Yes	Yes
Functional Separation	No	No	Yes	Yes
Require Middleware	No	No	Yes	No
High Fault-tolerance	No	No	No	Yes
decentralized governance	No	No	No	Yes
Low Resources Cost	No	No	No	Yes
Independent service deployment	No	No	No	Yes

From Table 2, we can exhibit that features of reusability, cross functional development, autonomous integration of services and resilience can be found in both SOA and microservices architectures. Though, scalability and loose coupling are only achievable with microservices. We can also notice that both SOA and microservices rely mostly on lightweight communication, yet REST APIs are mostly relied on in the microservices architecture which is lighter than other protocols used in SOA such as SOAP. In addition, SOA actually requires the existence of a middleware to interpret the incoming requests of unified protocols to lead to the suitable service for each client. On the other hand, microservices and Mobile Agents use Inter-Process communication. Yet the type of protocol used in microservices is simpler compared to Mobile Agents paradigm. In addition, both SOA and microservices are highly scalable, but scaling micro (small) services will take less time and resources than scaling a huge monolithic service in which service duplication is required.

Fewer features can be found in Client-Server and Mobile Agents, yet both architectures can be useful in applications which do not require scalability, resilience, service integration and other features found in either SOA or microservices. All architectures in fact are used for particular services types depending on their requirements priorities. Finally, when it comes to fault tolerance and low resources cost, microservices architecture only satisfies many of these features. However, they are only achievable by Client-Server, SOA and Mobile Agents when configured in ways supporting this matter. For fault tolerance, applications or services can be developed in order to overcome any faults by relying on more hardware resources. If fault tolerance was not as important as resources cost, then less resources can be used in which the application is built with minimum resources needed. So it is always a matter of priorities to be found in each application developed. Microservices architecture is not said to be the best among the others, but it actually can fulfil more features at once compared to the previously proposed architectures.

VII. CONCLUSION

Considering which architecture to use is always challenging. As explained in this paper, the reviewed architectures were made to fulfil several purposes depending on the demand of the services they are used for. Each architecture was explained in way leading to the next proposed architecture. In fact, it is not about following what is most modern and trendy, it is important to investigate whether the application would fully function efficiently using the selected architecture. Microservices advantages and drawbacks were illustrated to show whether the developed application would need that amount of scalability, integrity, resilience and agility with the payoffs of known and unknown consequences. A comparison was done in which the four illustrated architectures were compared in terms of the most desired features to be found in this modern time. We concluded that microservices are able to fulfil most features, yet they do come with many challenges. However, more studies around microservices should be done with consideration of their communication overhead, security and overall performance reflected on the application.

REFERENCES

[1] W. Lamersdorf, “Paradigms of distributed software systems: Services, processes and self-organization,” in International Conference on Ebusiness and Telecommunications. Springer, 2011, pp. 33–40.

[2] Akinnuwesi B. A (2011). Development of a user-centric model for evaluating distributed software system architecture using neuro-fuzzy technique. Ph.D. Thesis submitted to the Postgraduate school, Ladoke Akintola University of Technology, Ogbomosho, Nigeria.

[3] B. A. Akinnuwesi, F.-M. E. Uzoka, and A. O. Osamiluyi, “Neuro-fuzzy expert system for evaluating the performance of distributed software system architecture,” Expert Systems with Applications, vol. 40, no. 9, pp. 3313–3327, 2013.

[4] Mulesoft.Top Five Reasons Why APIs Are Taking Centre Stage for Bussiness. Software World Intelligence.

[5] L. Ismail, D. Hagimont, and J. Mossi`ere, "Evaluation of the mobile agents technology: Comparison with the client/server paradigm," Information Science and Technology (IST), vol. 19, 2000.

[6] W. A. De Vries and R. A. Fleck, "Client/server infrastructure: a case study in planning and conversion," Industrial Management & Data Systems, vol. 97, no. 6, pp. 222–232, 1997.

[7] C. Y. Subhash, Introduction to Client Server Computing. New Age International, 2009.

[8] A. A. Arokiaraj, "Efficient bandwidth utilization in client–server models," in Proceedings of the Third International Conference on Trends in Information, Telecommunication and Computing. Springer, 2013, pp. 563–570.

[9] A. Y. Gital, A. S. Ismail, H. Chiroma, A. I. Abubakar, B. M. Abdulhamid, I. Z. Maitama, and A. Zeki, "Performance analysis of cloud-based cve communication architecture in comparison with the traditional client server, p2p and hybrid models," in 2014 The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M). IEEE, 2014, pp. 1–6.

[10] C. Mantratzis and M. Orgun, "Towards a peer-to-peer world-wide-web for the broadband-enabled user community," in Proceedings of the 2004 ACM workshop on Next-generation residential broadband challenges. ACM, 2004, pp. 42–49.

[11] P. Apirajitha and A. Angayarkanni, "A design of an adaptive peer-to-peer network to reduce power consumption using cloud computing," in 2012 IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT). IEEE, 2012, pp. 198–201.

[12] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," in Proceedings of the tenth international conference on Information and knowledge management. ACM, 2001, pp. 310–317.

[13] M. Van Der Vlugt and S. Sambasivam, "Redesign of stand-alone applications into thin-client/server architecture," Informing Science: International Journal of an Emerging Transdiscipline, vol. 2, pp. 723–742, 2005.

[14] D. Iabbassen and S. Moussaoui, "Data dissemination protocols in wireless sensor networks client/server versus mobile agent paradigms," in Innovative Computing Technology (INTECH), 2015 Fifth International Conference on. IEEE, 2015, pp. 45–50.

[15] K. Kulesza, Z. Kotulski, and K. Kulesza, "On mobile agents resistance to traffic analysis," Electronic Notes in Theoretical Computer Science, vol. 142, pp. 181–193, 2006.

[16] P. Angin, B. Bhargava, and Z. Jin, "A self-cloning agents based model for high-performance mobile-cloud computing," in 2015 IEEE 8th International Conference on Cloud Computing. IEEE, 2015, pp. 301–308.

[17] Y. Mehmood, M. A. Shibli, A. Kanwal, and R. Masood, "Distributed intrusion detection system using mobile agents in cloud computing environment," in 2015 Conference on Information Assurance and Cyber Security (CIACS). IEEE, 2015, pp. 1–8.

[18] R. C. Cavalcante and A. L. Oliveira, "An autonomous trader agent for the stock market based on online sequential extreme learning machine ensemble," in 2014 International Joint Conference on Neural Networks (IJCNN). IEEE, 2014, pp. 1424–1431.

[19] S. Newman, Building Microservices. " O'Reilly Media, Inc.", 2015.

[20] Wen, Y. Ma, and X. Chen, "ESB infrastructure's autonomous mechanism of SOA," in 2009 International Symposium on Intelligent Ubiquitous Computing and Education. IEEE, 2009, pp. 13–17.

[21] P. Offermann, M. Hoffmann, and U. Bub, "Benefits of SOA: Evaluation of an implemented scenario against alternative architectures," in 2009 13th Enterprise Distributed Object Computing Conference Workshops. IEEE, 2009, pp. 352–359.

[22] B. Li, "Research and application of soa standards in the integration on web services," in 2010 Second International Workshop on Education Technology and Computer Science (ETCS), vol. 2. IEEE, 2010, pp. 492–495.

[23] N. Basias, M. Themistocleous, and V. Morabito, "An investigation of benefits affecting SOA adoption in e-banking," International Journal of e-Education, e-Business, e-Management and e-Learning, vol. 4, no. 3, p. 174, 2014

[24] V. Cardellini and S. Iannucci, "Designing a broker for QoS-driven runtime adaptation of SOA applications." in ICWS. Citeseer, 2010, pp. 504–511.

[25] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," International Journal of Open Information Technologies, vol. 2, no. 9, 2014.

[26] S. Alpers, C. Becker, A. Oberweis, and T. Schuster, "Microservice based tool support for business process modelling," in 2015 IEEE 19th International Enterprise Distributed Object Computing Workshop. IEEE, 2015, pp. 71–78.

[27] Y. Sun, S. Nanda, and T. Jaeger, "Security-as-a-service for Microservices based cloud applications," in 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2015, pp. 50–57.

[28] Hassan, M., Zhao, W., & Yang, J. (2010, July). Provisioning web services from resource constrained mobile devices. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on (pp. 490-497). IEEE.

[29] Namiot, D., & Sneps-Sneppe, M. (2014). On Micro-services Architecture. International Journal of Open Information Technologies, 2(9), 24-27.

[30] M. Vianden, H. Lichter, and A. Steffens, "Experience on a microservice based reference architecture for measurement systems," in 2014 21st Asia-Pacific Software Engineering Conference, vol. 1. IEEE, 2014, pp. 183–190.

[31] J. I. Fernandez Villamor, C. A. Iglesias Fernandez, and M. Garijo Ayestaran, "Microservices: Lightweight service descriptions for REST architectural style," 2010.

[32] B. Familiar, Microservices, IoT, and Azure. Springer, 2015.

[33] D. Malavalli and S. Sathappan, "Scalable microservice based architecture for enabling dmtf profiles," in Network and Service Management (CNSM), 2015 11th International Conference on. IEEE, 2015, pp. 428–432.

[34] V. D. Le, M. M. Neff, R. V. Stewart, R. Kelley, E. Fritzinger, S. M. Dascalu, and F. C. Harris, "Microservice-based architecture for the nrdc," in 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), IEEE, 2015, pp. 1659–1664.

[35] M. Rahman and J. Gao, "A reusable automated acceptance testing architecture for microservices in behavior-driven development," in Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on. IEEE, 2015, pp. 321–325.

[36] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to cloud-native architectures using microservices: An experience report," in European Conference on Service-Oriented and Cloud Computing. Springer, 2015, pp. 201–215.

[37] M. Vogler, J. M. Schleicher, C. Inzinger, and S. Dustdar, "A scalable framework for provisioning large-scale IoT deployments," ACM Transactions on Internet Technology (TOIT), vol. 16, no. 2, p. 11, 2016.

[38] E. Lacic, M. Traub, D. Kowald, and E. Lex, "Scar: Towards a real-time recommender framework following the microservices architecture." 9th ACM Conference on Recommender Systems, Sep 2015, Vienna.

[39] D. McCarthy, P. Malone, J. Hange, K. Doyle, E. Robson, D. Conway, S. Ivanov, Radziwonowicz, R. Kleinfeld, T. Michalareas et al., "Personal cloudlets: implementing a user-centric datastore with privacy aware access control for cloud-based data platforms," in Proceedings of the First International Workshop on TEchnical and LEgal aspects of data pRIvacy. IEEE Press, 2015, pp. 38–43.

[40] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure DevOps," in 2016 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2016, pp. 202–211.

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

[42] J. Stubbs, W. Moreira, and R. Dooley, "Distributed systems of microservices using docker and serfnode," in Science Gateways (IWSG), 2015 7th International Workshop on. IEEE, 2015, pp. 34–39.

[43] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a smart city internet of things platform with microservice architecture," in Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on. IEEE, 2015, pp. 25–30.

[44] M. Villamizar, O. Garc´es, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in Computing Colombian Conference (10CCC), 2015 10th. IEEE, 2015, pp. 583–590.

[45] M. E. Br¨uggemann, R. Vallon, A. Parlak, and T. Grechenig, "Modelling microservices in email-marketing concepts, implementation and experiences," in 9th International Conference on Software Paradigm Trends (ICSOFT-PT). IEEE, 2014, pp. 67–71.

[46] "Ebay," https://www.ebay.com/.

[47] P. Nicolaescu, G. Toubekis, and R. Klamma, "A microservice approach for near real-time collaborative 3d objects annotation on the web," in International Conference on Web-Based Learning. Springer, 2015, pp. 187–196.

[48] D. Guo, W. Wang, G. Zeng, and Z. Wei, "Microservices architecture based cloudware deployment platform for service computing," in 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE). IEEE, 2016, pp. 358–363.

[49] Y. Kato, T. Izui, Y. Murakawa, K. Okabayashi, M. Ueki, Y. Tsuchiya, and M. Narita, "Research and development environments for robot services and its implementation," in System Integration (SII), 2011 IEEE/SICE International Symposium on. IEEE, 2011, pp. 306–311.

[50] P. Bak, R. Melamed, D. Moshkovich, Y. Nardi, H. Ship, and A. Yaeli, "Location and context-based microservices for mobile and internet of things workloads," in 2015 IEEE International Conference on Mobile Services, 2015, pp. 1–8.

[51] D. Savchenko, G. Radchenko, and O. Taipale, "Microservices validation: Mjolnirr platform case study," in Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on. IEEE, 2015, pp. 235–240.

[52] S. Patanjali, B. Truninger, P. Harsh, and T. M. Bohnert, "Cyclops: a micro service based approach for dynamic rating, charging & billing for cloud," in Telecommunications (ConTEL), 2015 13th International Conference on. IEEE, 2015, pp. 1–8.

[53] OMG. The Common Object Request Broker: Architecture and Specification, OMG Document Number 91.12.1, Revision 1.1, December 1991.

[54] L. Florio, "Decentralized self-adaptation in large-scale distributed systems," in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015, pp. 1022–1025.

[55] A. Levcovitz, R. Terra, and M. T. Valente, "Towards a technique for extracting Microservices from monolithic enterprise systems," arXiv preprint arXiv: 1605.03175, 2016.

[56] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. , F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani et al., "IoT design patterns: Computational constructs to design, build and engineer edge applications," in 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI). IEEE, 2016, pp. 277–282.

[57] Canonical. Linux Containers. 2015. https://linuxcontainers.org/ (visited on 01/06/2016).

[58] A. Grillet, "Hot topics in ISE - topic 5 comparison of container schedulers." http://armand.gr/static/files/htise.pdf.

[59] "Dies", http://deis.io/.