Towards Reference Architecture for a Multi-layer Controlled Self-adaptive Microservice System

Peini Liu, Xinjun Mao, Shuai Zhang, Fu Hou
College of Computer
National University of Defense Technology
Hunan, China 410073
Email: peini.liu@foxmail.com, xjmao@nudt.edu.cn, zhangshuai16a@nudt.edu.cn, houfu@nudt.edu.cn

Abstract—**With the features of high distribution in deployment and independence in running, the microservice systems that operate in heterogeneous infrastructures and open Internet environment are expected to be self-adaptive to adapt to various changes of both operating contexts and application requirements. This requires the adaptability of the microservice systems to be diverse and flexible, and independent of implementation technologies and platforms. This paper presents a reference architecture for self-adaptive microservice systems with the abilities of multi-layer controlled self-adaptations, including infrastructure-controlled layer and application-controlled layer. Such reference architecture presents a blueprint to cope with diverse changes from different levels in microservice systems and supports the interactions between layers. We have implemented a practical platform called *SAMSP* based on the reference architecture and Kubernetes and evaluated our approach using a sample. The experimental results are promising, and demonstrate the feasibility and effectiveness of our proposed reference architecture.**

Keywords-microservice system; reference architecture; self-adaptive microservice system; multi-layer control loops

I. INTRODUCTION

Microservice, a popular architectural style, attracting more and more attention in both academia and industry areas, is widely adopted now by many large companies such as Amazon [1], Netflix [2], LinkedIn [3]. This architecture style is considered to the best efforts for cloud computing and service-oriented system engineering [4].

In the early years of service-oriented architecture, the monolithic architectural style has been an approach to build web applications. These applications were built as a single unit, and all the logic for handling a request runs in a single process [5]. As system under this architecture is lack of independence and flexibility, services have to get scaled and evolved together, which result in a huge waste of server resources. Hence, the monolithic architectural style is not suitable enough to construct ultra-large-scale information system anymore. To overcome the challenges, microservice has become a new architectural style to build such complex system [6]. It decomposes a large complex software application into a suite of small services, with each service running in its own process and communicating by lightweight mechanisms [7]. The microservice architecture style brings many benefits for service-oriented systems, such as scalability [8], functional separation [9], loose coupling [7] and fast delivery [10].

However, microservice system still faces serval challenges. Firstly when turning into microservice, since these highly distributed microservices often run on containers deployed on cloud and are organized to realize an application, we have to take a software architecture with adaptability. Secondly, as the microservice system is evolving because of changes of context and requirements, system must adapt to handle the challenges. In addition, the agile and DevOps methodology expect the system runs without downtime and integrates continuously. Therefore, system is no longer running in a known context and with static requirements, meaning that the system needs to reconfigure and restructure themselves to meet the dynamic changing world [11].

Obviously, it is infeasible for operators to take all the changes into consideration and manually control such system. However, self-adaptive system brings some inspiring approaches to adapt system at runtime, which help to preserve and optimize the system's operation in dynamic changes [12, 13]. For example, Rainbow [14], an architecture-based approach, allows the self-adaptive system to be aware of the software structure and drives the self-adaptation by the external control. Another reflection approach can use the reflected ability of software to examine and possibly modify its structure (structural reflection) or behavior (behavioral reflection) at runtime [15]. These two methods inspire us to build MAPE control loops as an autonomic manager to monitor the states of microservice system, to analyze the changing and to plan and execute the actions at runtime [16].

Our work presents a reference architecture for a multi-layer controlled self-adaptive microservice system. The self-adaptation idea comes from self-adaptive systems which can manage system itself according to the high-level goals [16]. The reference architecture makes microservice system to be aware of its dynamic contexts continuously and the changing requirements in order to adjust its behavior and structure at runtime. The innovation of this architecture is using autonomic computing MAPE control loop to form multi-layer control loops. On the one hand, self-adaptive microservice system might run on third-party provided infrastructure servers, and the system workload at the infrastructure layer has to be adapted through infrastructure-controlled loop. On the other hand, self-adaptive

DOI reference number: 10.18293/SEKE2018-086

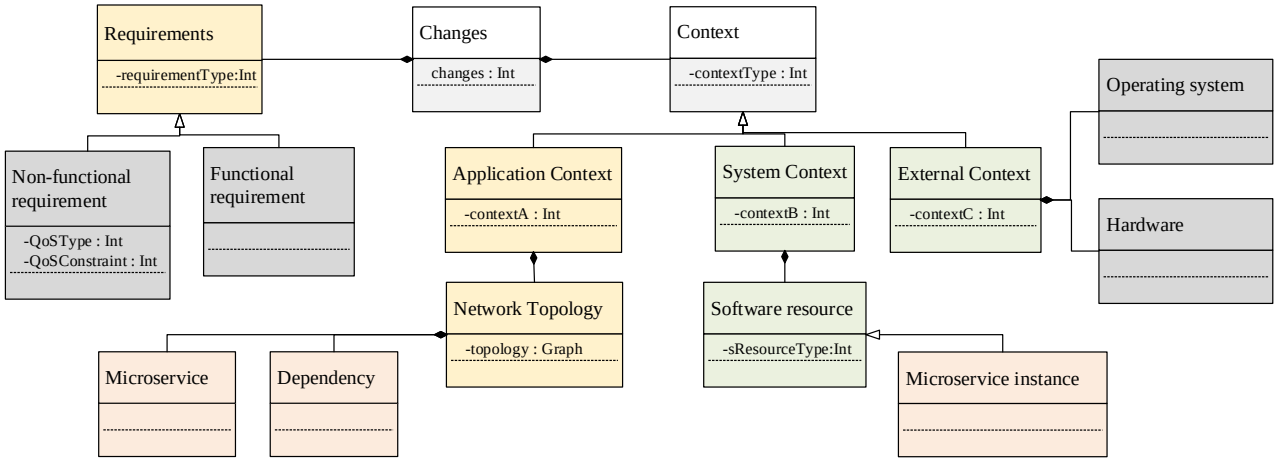

Figure 1. Diversity of changes in self-adaptive microservice system.

microservice system has multiple microservices organized to realize an application, and the applications' requirements and performance at the application layer have to be assured through application-controlled loop.

The remainder of this paper is organized as follows: Section 2 analyses the diverse changes and multiple layers in self-adaptive microservice system. Section 3 proposes a reference architecture for a multi-layer controlled self-adaptive microservice system; Section 4 presents a practical implementation of the reference architecture; a sample and a promising experiment based on reference architecture are given in Section 5; Section 6 compares with related work and section 7 concludes and discusses the future work.

II. Diverse Changes and Multiple Layers in Self-adaptive Microservice Systems

Self-adaptive microservice system has challenges to facing the diverse changes. Inspired by Cherif Sihem et al.[17] who bring a context model of SOA that divided into several parts—Infrastructure, Platform, and Application. Figure 1 represents the diversity of changes in self-adaptive microservice system.

Changes in self-adaptive microservice system contain the context and the application requirements, each of which has its own rules and presents a part of self-adaptation. The former changes have three basic elements that provide different levels of context from different perspectives. For example, (1) external context is a part of the external world and includes hardware resource, operating system and other related system. System can sense these context but cannot directly control. (2) system context is internal system environment with software system resources like microservice instance. (3) application context includes system network topology, and they are concerned with specific application. The latter changes are from users and include application functional and non-functional requirements. For instance, (1) functional requirements are about the organization of applications. (2) non-functional requirements are the QoS related to the application performance.

Each part of changes has its own adaptation rules that can be conducted. However, from the result of adaptation, the system needs to achieve adaptation by reconfiguring microservice instances or restructuring the microservice topology. So we can provide adaptation facilities isolated in the infrastructure and application two layers. Thus, a key challenge is how to structure and coordinate the two layers to handle the diverse changes.

To overcome the challenge, the structure with two control layers upon the microservice system is shown, and a conceptual model of multi-layer controlled self-adaptive microservice system is proposed in order to clarify the layers and their interactions. In Figure 2, the self-adaptive microservice system is composed of multi-layered control layers and the microservice system. Microservice system is a target system that achieves the function of business. Managing system has multiple controlled layers, and each of the layer manages different types of adaptation: (1) the infrastructure-controlled layer (ICL) senses the system context and external context to manage the containers with the platform predefined rules, system adaptation often appears as reconfiguration; (2) the application-controlled layer (ACL) senses the application related changes like requirements and application context, system adaptation can be restructured or with the lower control loops help.

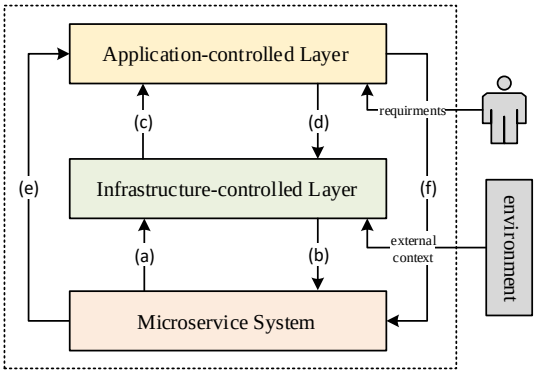

Figure 2. The conceptual model of multi-layer controlled self-adaptive microservice system.

III. Reference Architecture for Self-adaptive Microservice Systems

Nowadays, microservice system needs a reference to support continuous changes in context and requirements. Facing the challenge in Section II, in this section we proposed a reference architecture for self-adaptive microservice system as a guideline

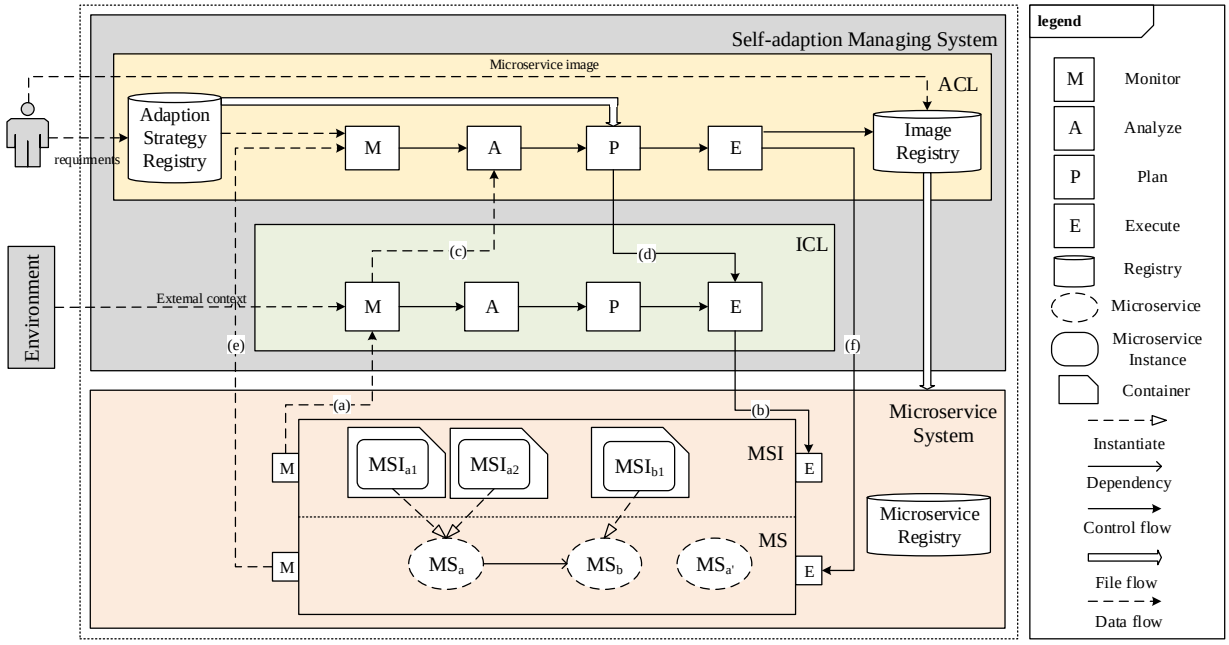

Figure 3. The reference architecture for multi-layer controlled self-adaptive microservice system.

for engineers on how to design, develop and adapt such systems in a specific domain. Our reference architecture is inspired by the control theory and find an effective instantiation –'MAPE model' to build control loops. In fact, our contribution is to design the architecture by considering these aspects explicitly: (1) decoupling the different layer control loops required to satisfy each part of changes; (2) achieving self-adaptation goals through the collaboration between two layers.

A detailed view of the reference architecture for multi-layer controlled self-adaptive microservice system is given in Figure 3. The reference architecture defines the functional elements, as well as the control, data or file interactions among the internal elements of each layer and the multiple layers. In addition, the reference architecture characterizes the collaboration among the multi-layer to assure that it can be applied partially in case someone does not need some parts of adaptation. These details are explained in the following sections.

A. *Microservice system*

As a target system, the microservice system consists of a set of microservices, which are organized in applications through lightweight protocols. There are two concepts in a microservice system: one is a microservice instance (MSI), which refers to a real entity that handles requests to accomplish the appropriate functions. The other is a microservice (MS), which can be understood as an abstraction of a set of microservice instances which exactly have the same capabilities.

At run time, the microservices are discovered by each other through the microservice registry. In particular, the microservice itself does not process the request but distribute the request to its corresponding microservice instances to perform the functions. The microservices instance is the smallest running unit that runs in a container and is deployed on cloud, giving us an inspiration to operate the container to manage microservices instances. Meanwhile, microservices also shield the operation details through the abstract microservices interface and well maintain the topology of the application. Once the topology changes due to the change of the dynamic context or the change of applications' requirements, the adaptive system can timely observe the microservices and dependencies between them to restructure the application. Therefore, the microservice system needs to sense the running status information of the microservice instance and the topology of the organized microservice application to determine whether the target system is healthy.

B. *Infrastructure-controlled layer*

The infrastructure-controlled layer (cf. ICL in Figure 3) solves the adaptive problem at the infrastructure level of the adaptive microservice system. It consists of a MAPE control loop: the *Monitor* senses the external context from the environment and the system context from the microservices instances (cf. Interaction (a) in Figure 3) and collects monitoring data. *Analyze* analyzes the system-related information, and triggers the system-level policy in *plan* by the event whether the context changes. Finally, *Execute* in the control loop adjusts the system configuration according to the policy, so that the system adaptation can be implemented by scheduling the place that containers deployed, scaling the number of containers and limiting or increasing the resources of the container (cf. Interaction (b) in Figure 3).

C. *Application-controlled layer*

The application-controlled layer (cf. ACL in Figure 3) as the upper layer of self-adaptive microservices system, also consists of a MAPE control loop: *Monitor* senses the context from the application organized by the microservices — the application topology (cf. Interaction (e) in Figure 3) or the requirement changes by users through adaptation strategy, and *Analyze* analyzes the application functional requirements information and triggers the strategy written by the application developer in *Plan* when the topology changes, e.g., changing the dependency

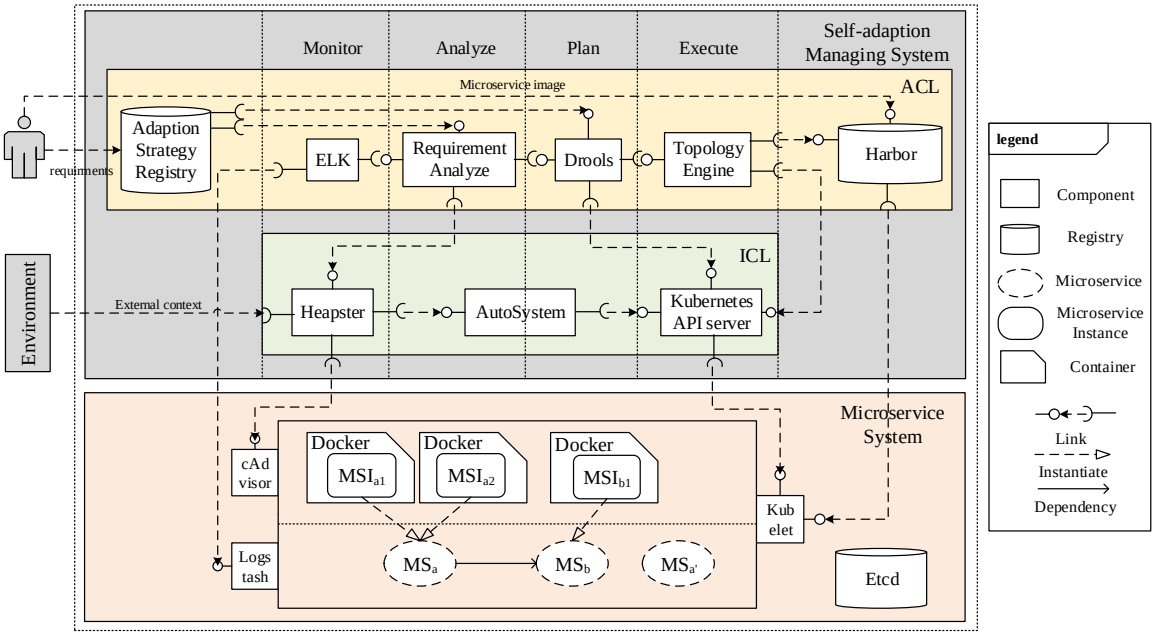

Figure 4. *SAMSP* - A practical implementation of the reference architecture.

between two microservices. Finally, *Execute* in the loop achieves the system adaptation according to the tactics that adjust the microservices organized in application and their related dependency (cf. Interaction (f) in Figure 3).

Moreover, ACL governs a part of changes with the collaboration of the ICL. We define the application-related runtime information at the ICL layer as variables to be controlled in ACL. In the circumstances, ICL *Monitor* uploads the related data to ACL *Analyze* (cf. Interaction (c) in Figure 3) to judge whether it satisfies the application non-functional requirement and decide to obtain the results in *Plan*. At last, ACL *Plan* sends the approach to ICL *Execute* (cf. Interaction (d) in Figure 3) to mentor the runtime adaptation.

IV. A Practical Implementation of the Reference Architecture

This section describes a practical implementation of the reference architecture that is built based on Kubernetes[1]. Our implementation provides an extended platform called *SAMSP* with a toolkit to realize the multi-layer self-adaptation. The whole implementation architecture of the reference architecture is depicted in Figure 4.

In the design stage, application logic and adaptive logic are separated. For one, we develop microservices with independent function and use jersey.jar to implement the Restful interaction protocol. After the development, we structure their environment through Docker[2] images and put them into Image registry Harbor[3]. For the other, some self-adaptation goals are considered by using self-adaptation strategy language to describe and registered in our adaptation strategy registry.

When it comes to runtime stage, in microservice system, microservice instantiate several microservice instances, running in containers and deployed on distributed cloud servers. We use a container orchestration Kubernetes to help us deploy our microservice instances containers, also, some plugins like Etcd[4] which is used for microservice discovery. As an important role in monitoring system status, cAdvisor[2] collects the performance of the microservice instances, and the logstach obtains a calling chain in local.

In ICL, in the monitor stage, we use Heapster[5] to collect the status of clusters and microservices instances' performance like CPU usage, memory usage, etc. in local cAdvisor[2]. In analyze and plan stages, the Autosystem component that expanded from HPA (Horizontal pod autoscaler) in Kubernetes to analyze the status of the system and choose optimal values for the configurable parameters. In the execute stage, the Kubernetes API server hands out the new configuration parameters to the cluster kubelet to adapt the changing context.

In ACL, firstly, ACL needs to load the adaptation strategies from the adaptation strategy registry. In the control loop, internally, ELK[6] (ElasticSearch, Logstash, Kibana) are used to collect the organization of the application from local logstash. After the requirement check from requirement analyze, Drools[7] as our application rules engine will fire the self-adaptation strategy we have defined and use topology engine to change the dependency between the microservices or build a new structure. As for collaborating with ICL, the requirement analyze asks the application-related property from ICL and the Kubernetes API server obtains an application required results from Drools to operate the containers.

V. Case Study and Experiments

To illustrate the feasibility of our proposed reference architecture and the effectiveness of its self-adaptation, in this section, we use a book information system (BIS) as a running

[1] https://kubernetes.io/
[2] https://www.docker.com/
[3] http://vmware.github.io/harbor/
[4] https://coreos.com/etcd/
[5] https://github.com/kubernetes/heapster/
[6] https://www.elastic.co/products
[7] https://www.drools.org/

example [18]. This application provides the book information support to help users to know the book through the internet. Basically, the BIS application shows the information of books by composing a book review microservice with a book content microservice. Meanwhile, the book contents can be provided by some different book content information providers, such as Wikipedia, Baidupedia and several school library systems.

A. Sample development based on BIS

Here, the generic adaptation scenarios for microservice system are divided in two aspects, the infrastructure level adaptation and the application level adaptation. (see Table 1)

TABLE I. GENERIC ADAPTATION SCENARIOS OF MICROSERVICE SYSTEM

Layer	Type of changes	Scenarios
ICL	*External context*	*S1: Microservice instance unavaliable*
	System context	*S2: Microservice instance overload*
ACL	*Application context*	*S3: Microservice unreachable*
	Functional Requirements	*S4: Application function enhancement*
	Non-functional Requirements	*S5: Application-related QoS constraints violation*

A real BIS application has been set up based on our reference architecture. This BIS adopts microservice, and the ICL/ACL can be implemented exactly as Section IV. The ICL layer is used to enhance the reliability and the performance of the system by reconfiguring the microservice instances. It takes effect on S1 and S2 in Table 1. In S1, if one of these book review microservice instances is unavailable, the rest of the book review microservice instances need to accept the requests from the unavailable microservice instance, and to restart this failed microservice instance. (2) S2: if one of them is overloaded, reconfiguring the number of the microservice instances or scheduling it to an available server will achieve better system performance.

However, if all instances failed, the microservice will be unreachable. So the ACL is responsible to handle this through restructuring the organization of the application. For instance, in S3, if the book content microservice is unreachable, the ACL will be notified the situation. In response, this layer will register the alternative service (e.g. Wikipedia or Baidupedia) into the system, adjust the dependency between these related microservices and relink that service to the latest available microservices. For requirement changes, in S4, if the application needs a new function, e.g., book rating, ACL will handle the functional requirements by pushing a new microservice into image registry and a new configuration into the ICL to make use of the microservice that is newly added. Finally, as for the collaboration between the ACL and ICL, the scenario is that, if the user requires the average response time of the service to be no more than 1.5s (i.e. S5). To satisfy this, ACL will sense the average response time information from ICL and trade-off the planning when the condition has been violated, at last execute the plan in ICL.

B. Experiments Analysis

To evaluate the self-adaptation effect and performance in the self-adaptive microservice system compared to the original microservice system without self-adaptation, we conduct an experiment on a distributed testbed Locust[8]. In this experiment, some of the adaptation scenarios from Table 1 have been selected (e.g. S1: Microservice instance failure, S2: Microservice instance overload, S5: Application-related QoS constraints violation). We observe the microservice's average response time (ART) in 30min to evaluate the performance when microservice faces diverse changes.

Figure 5 shows the results of system performance with and without adaptation during the whole running time. The solid line shows that, without adaptation, once the average response time rises when facing the changes, it never falls again. On the other hand, the dashed line shows that if *SAMSP* works in the adaptation, average response time rises but soon return to optimal level. Table 2 shows the details of the changes in the periods, and also compared the results of our case. The results indicate that self-adaptation is effective and significantly improves system performance in this experiment.

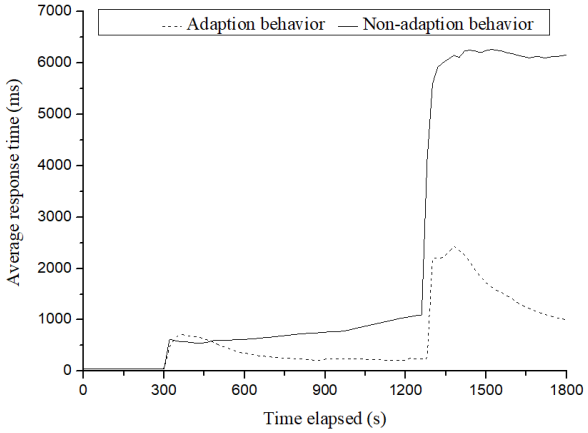

Figure 5. System performance with and without adaptation.

TABLE II. SCENARIOS PERIOD AND RESULTS OF THE CASE STUDY

Time (s)	Scenarios	ART without adaptation(ms)	ART with adaptation(ms)
0-300	*System stable running*	51.11	47.15
301-900	*S2: Book information microservice overload*	644.81	408.32
901-1200	*S1: Book information microservice instance failure*	895.51	231.26
1201-1800	*S5: Book information microservice QoS constraints violation*	5559.90	1454.25
Total time		2148.94	646.67

VI. RELATED WORK

We find some work related to self-adaptive microservice system, and also we take a look at some models for designing a self-adaptive system and the microservice architecture nowadays. Some work is briefly described in this section.

[8] https://www.locust.io/

An architecture for self-managing microservices is presented at [19]. It proposed a novel architecture that enables scalable and resilient self-management of microservices application on cloud. The main approach is using the algorithm to select a leader to assign management functionality to nodes and allowing atomic service to become self-managing. However, the distributed configuration management is much easier to conflict compared to centralized architecture and cannot implement expensive algorithms to elect leaders. A reference architecture from Krasimir based on SOA and autonomic computing [20] provides a way to transform the microservice instances by adding an autonomic manager into a part of the service and build an adaptation registry. However, in this approach, microservice instances need to be transformed into an intelligent instance by weaving code, which is quite difficult and expensive to implement. In [21], Namiot provides an overview of microservices architecture and implementation pattern. However, it treats microservices as components and analyses its communications in a design view. An autonomic computing system supports a continuous process, J. Kephart in [22] discuss a view of autonomic computing that has four elements: Monitor, Analyze, Plan and Execute (MAPE). This model reflects a general method for self-adaptive system.

Our research is different from previous work among the following: (1) We analysis the diversity of changes and divide adaptation into different layers; (2) We present a reference architecture for self-adaptive microservice system with the multi-layer control loops; (3) Our research ensures the coherence between the reference architecture and implementation and deploys a case in real system.

VII. Conclusion and Future Work

Microservice system is distributed and deployed on the cloud, which often uses container technology. It needs the capability of self-adaptation to face the diverse changes and challenges. Our contribution is discussing this question from an architecture perspective, and presenting a reference architecture for a multi-layer controlled self-adaptive microservice system, which constitutes a guide to design self-adaptive microservice systems.

Our contributions are threefold: (1) designing a novel reference architecture for multi-layer controlled self-adaptive microservice system, which decouples different layer control loops required to satisfy different parts of changes, and achieves self-adaptation goals through reconfiguring or restructuring the microservice system at runtime; (2) presenting an implementation architecture of microservice systems based on our reference architecture and K8S, and providing an extended platform *SAMSP* with a toolkit; (3) validating our proposed reference architecture in term of sample development and experiments, and showing the feasibility and effectiveness of architecture and platform for self-adaptive microservice systems.

For further research, we would like to improve our reference architecture in the following aspects: (1) enhance the self-adaptation managing system to provide more common self-adaptive abilities; (2) design a context model and relation model for microservice systems as self-adaptation knowledge to improve the control loop; (3) integrate accurate and efficient algorithms to choose adaptation strategies optimally.

Acknowledgement

This research is supported by research grants from Natural Science Foundation of China under Grant No. 61532004 and 61379051.

References

[1] Staci kramer. gigaom - the biggest thing amazon got right: The platform. https://gigaom.com/2011/10/12/419-thebiggest-thing-amazon-got-right-the-platform/, 2011.

[2] Tony mauro. nginx - adopting microservices at netflix: Lessons for architectural design. http://nginx.com/blog/microservices-at-netflix-architecturalbestpractices/, 2015.

[3] Steven ihde. infoq - from a monolith to microservices + rest: the evolution of linkedin's service architecture. http://www.infoq.com/presentations/linkedin-microservices-urn, 2015

[4] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. IEEE Computer Society Press, 2016.

[5] M Fowler, j lewis. monolith first. http://martinfowler.com/bliki/MonolithFirst.html, 2015

[6] Mario Villamizar, Oscar Garcs, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In Computing Colombian Conference, 2015.

[7] M fowler, j lewis. microservice a definition of this new architectural term. https://martinfowler.com/articles/microservices.html, 2015.

[8] Thomas F. Dllmann and Andr Van Hoorn. Model-driven generation of microservice architectures for benchmarking performance and resilience engineering approaches. In The Acm/spec, pages 171–172, 2017.

[9] Sara Hassan, Nour Ali, and Rami Bahsoon. Microservice ambients: An architectural meta-modelling approach for microservice granularity, 04 2017.

[10] Tasneem Salah, M. Jamal Zemerly, Yeob Yeun Chan, Mahmoud AlQutayri, and Yousof Al-Hammadi. The evolution of distributed systems towards microservices architecture. In Internet Technology and Secured Transactions, pages 318–325, 2017.

[11] Schmerl B, Kazman R, Ali N, et al. Managing Trade-Offs in Adaptable Software Architectures. 2017.

[12] Danny Weyns. Software engineering of self-adaptive systems: an organised tour and future challenges. 2017.

[13] Krupitzer C, Roth F M, Vansyckel S, et al. A survey on engineering approaches for self-adaptive systems. Pervasive & Mobile Computing, 17(PB):184-206, 2015.

[14] Cheng Huang, David Garlan, and Bradley Schmerl. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer, 37(10):46–54, 2004.

[15] J. Malenfant, M. Jacques, and F. N Demers. A tutorial on behavioral reflection and its implementation. 1996.

[16] IBM corp. an architectural blueprint for autonomic computing, tech. rep. http://www03.ibm.com/autonomic/pdfs/AC, 2005

[17] Cherif S, Djemaa R B, Amous I. ReMoSSA: Reference Model for Specification of Self-adaptive Service-Oriented-Architecture. ADBIS. p121-128, 2014.

[18] Danny Weyns and Radu Calinescu. Tele assistance: A self-adaptive service-based system exemplar. 05 2015.

[19] Giovanni Toffetti, Sandro Brunner, Florian Dudouet, and Andrew Edmonds. Anarchitecture for self-managing microservices. InInternational Workshop on Automated Incident Management in Cloud, pages 19–24, 2015.

[20] Krasimir Baylov and Aleksandar Dimov. Reference architecture for self-adaptive microservice systems. pages 297–303, 2017.

[21] Dmitry Namiot and Manfred sneps sneppe. On micro-services architecture. 2:24–27, 09 2014.

[22] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. Computer, 36(1):41–50, January 2003