A Design with Mobile Agent Architecture for Refactoring A Monolithic Service into Microservices

Masayuki Higashino*, Toshiya Kawato, Takao Kawamura
Tottori University, Tottori, 680-8550, Japan

* Corresponding author. Tel.: +81+0857-31-6100; email: higashino@tottori-u.ac.jp
Manuscript submitted June 5, 2018; accepted August 20, 2018.
doi: 10.17706/jcp.13.10.1192-1201

Abstract: Refactoring monolithic architecture into microservice architecture is a difficult task. In many cases, a monolithic service is divided into N-tiers based on the N-tier architecture. In order to divide a monolithic service into microservices, it is necessary to redefine a model as a new microservice by extracting models across layers and integrating them. However, since different layers and architectures are used for each layer, such as a database, an application framework, server software, etc., programs and models extracted from each tier are often redesigned and re-implemented in many cases. In this paper, we focus to the mobile agent technology that builds a system only by the simple two methods of agent's migration between computers and messaging between agents; and we propose the system architecture to facilitate the migration of a monolithic service to microservices.

Key words: Distributed system, microservice, mobile agent, monolithic service, web service.

1. Introduction

With the evolution of cloud computing and web technology, many web services have been developed and operated which are updated frequently and continuously. Typically, many web services are built on a multi-layered architecture, and each layer has a monolithic architecture. However, the internal implementation of these layers is getting more complicated, and a wide area of rebuilds and redeployment may be required for updating the system. Operation of a system requiring frequent and continuous change requires a wide area of changes to be a burden of development and operation.

For this reason, it is necessary to localize the area of influence on the changed software module.

Against this background, the usefulness of microservice architecture, which constructs a system by combining software components of microservices, is being evaluated [1], [2].

There is currently no definition for microservices. According to [3], a single application is developed by combining multiple microservices. These microservices run in their processes and often communicate with each other by a lightweight communication protocol such as the REST API (representational state transfer application programming interface) [4]. Also, these microservices are built on the business capabilities and can be deployed independently by fully automated mechanisms. Centralized management for those services is minimal, and each service can use different programming languages and different data storage technologies.

The features of such a microservice architecture are common to mobile agent technology, which are autonomous software components that can migrate among computers connected to the network, and are thought to have high affinity with each other.

A mobile agent is an autonomous software module which can migrate between different computers via computer networks. A paradigm and a behavior of mobile agents are designed like humans and whose society such as collaboration and competition among people. This feature of mobile agents is assumed to contribute to the ease of management for microservices because microservices are hard to be managed these life cycle and relations among them as a distributed dynamic software module. The problem area of microservice architecture partly overlaps with mobile agent technology.

This paper discusses an design for application of mobile agent technology to microservice architecture and shows requirements and design for a mobile agent framework to manage microservices on the web.

2. Requirements of Architecture

In general, web services are not designed with microservice architecture at the beginning of launching. Monolithic architectures and frameworks such as Ruby on Rails [5], CakePHP [6], etc. with high development efficiency are used early in many web services. It is difficult to predict in advance what feature is demanded at the time of launching the web service, making microservice at the time when necessity does not occur becomes a factor to raise the development cost of the system. Basically, disassembling software into many parts like a microservice is a trade-off between system complexity and maintainability, so it is common to adopt a monolithic architecture unless it is necessary. Thus, it can be said that one of the essential problems of microservice is a realization of easy change from monolithic architecture to microservice architecture.

In the following sections, we discuss requirements for simplifying changes from a monolithic architecture to microservice architecture using the paradigm of the mobile agent system.

2.1. Data Distribution

In the case of developing a web service, in recent years, relational databases, document databases, graph databases, key-value stores and the like are often adopted.

What is a problem when distributing these data to multiple computers (services) is to ensure data consistency. Generally, data stored in a logical or physically separate computer has a lower cost of ensuring consistency because latency and throughput of the network between computers are lower as the data is stored. Thus, the programmers of the system examine which data is strongly or weakly related to each other, and adopts an approach in which the relationship between data is weak, or those with a low frequency of association are preferentially distributed. This approach is often adopted in the database architecture called NoSQL [7].

It is not unusual for the structure of data to change in the operation of the web service. In the microservice architecture, to localize the influence area of the maintenance of the system, an approach is taken to lower the degree of coupling between the services and increase the degree of condensation of the service. This is very similar to the optimization problem of the consistency assurance level strength and cost associated with the horizontal distribution of the database. In other words, it is thought that it is highly compatible with the architecture of microservices that localize the scope of influence of maintenance related to the service by putting it in a location that is closer to the relevant data.

Therefore, we propose to extract a data model with a strong association strength between models as a microservice while measuring the frequency of transactions of data generated by multiple data models and the communications traffic of data.

In addition, by enabling the operation to easily extract the microservice, it is thought that it becomes easy to change the monolithic web service to the microservice architecture.

A mobile agent is a useful aspect in cases where autonomy and transfer between computers are required.

By providing a framework consisting of a suitable programming language and execution environment for this purpose, it is considered that the scalability of the system that can be easily realized can be achieved.

2.2. Process Distribution

To process distribution, those with less data input/output can be distributed relatively easily. On the other hand, a computer responsible for a large amount of data processing is strongly related to the data consistency guarantee mentioned above, so basically it is necessary to locate it close to the computer where the data is stored.

Therefore, the microservice having the function of processing data has the greatest influence on the network distance between the microservice having the data to be processed, the microservice as the processing result output object, and the end user's computers It is required to migrate to a microservice with low cost.

Such a requirement is a field that the mobile agent is good at, and it seems that affinity between autonomous processing dispersion and mobile agent is high.

2.3. Domain-Driven Design

When dividing an existing service into microservice, a method of searching for a junction that can achieve both loose couplings and high condensability in the problem domain is used [1], [8].

However, this junction may go beyond the technical boundary. For example, in web services, a three-layer architecture is often adopted, but junctions that can achieve both loose coupling in the dispersion of data and dispersion of processing and high condensability traverse these layers there is a possibility.

Therefore, it is difficult to adopt the existing monolithic web application framework as it is. Many of these frameworks clearly divide user interface functions, data processing functions, and data storage functions into layers.

For this reason, by adopting a mobile agent, which is a small software component capable of both data retention and data processing, as a system platform, it is possible to realize a junction capable of achieving both loose couplings in the problem domain and high condensability It is thought that it can be flexible and easily extracted as a microservice.

Thus, we propose an approach that extracting microservices from a monolithic service and reconstructing a monolithic service to microservices dynamically.

3. Requirements of Framework

The smaller the service, the higher the cost for grasping the specification of each service and applying it accurately. For this reason, as compared to performing static specification verification in advance, the importance of being able to dynamically respond to changes in the system is increased when the operation of each microservice is changed.

Therefore, we propose a method to dynamically extract microservices from a monolithic service and dynamically reconstruct them as microservices. There are two important properties to realize this proposal.

3.1. Framework for Re-construction

First, any part of a monolithic service or a microservice can be divided as a new microservice, and it works properly as a system. This can be realized by utilizing many existing virtualization technologies that have been proposed. However, the granularity of virtualization ranges from function of programming language to virtual machine. It is considered that mobility by mobile agent technology and correspondence to dynamic constraint satisfaction problem can be utilized for this.

Because relaxing restrictions on models based on business logic that developers think when dividing as microservices, and models based on hardware and technology being used, we can redefine from a more flexible monolithic service to microservice It is because our proposal intends to make the configuration possible.

For this reason, it is not an existing web framework based on restrictions of technical specifications such as a three-layer architecture and a request-response type such as an HTTP [9], rather than being as conscious as possible of restrictions on technical specifications, a more flexible monolithic service A new web framework with constraints and conventions that make it easier to split into microservices.

3.2. Evaluation of Cost with Re-construction

The second is to be able to evaluate the cost of extracting new microservice from a monolithic service. Even though it is possible to divide the system, the performance of the system depends on constraints of the underlying hardware and software, so even if an advantage as microservices is obtained by division, estimation of performance degradation by division and measures to prevent irreversible division are extremely important.

Therefore, our proposal is based on the collection and analysis of statistical information on the inside of the service, based on the change of the performance cost when dividing microservices from a monolith service and the fact that it can be merged to the original system after being divided, In other words, it is necessary to think about a method to evaluate reversibility.

In our future work, we will proceed with these two studies.

4. Design for Architecture

4.1. Mobile Agent

First of all, we define an internal data structure of a mobile agent. Fig. 1 shows the architecture of a mobile agent system. In AREs (agent runtime environment), multiple agents work concurrently and are able to migrate between AREs via networks. An agent is constructed from a runtime state area and an application area, and a program code area that contains program codes [10]. A runtime state area has information of states of the agent during executions of tasks such as call stacks, program counters, etc. An application area has any data that are specified by developers of the agent. A program code area has a set of program codes that are described behaviors of the agent. This constructions cover specifications of MASIF Specification by OMG (Object Management Group) [11] and Agent Management Specification by FIPA (Foundation for Intelligent Physical Agents) [12], [13].

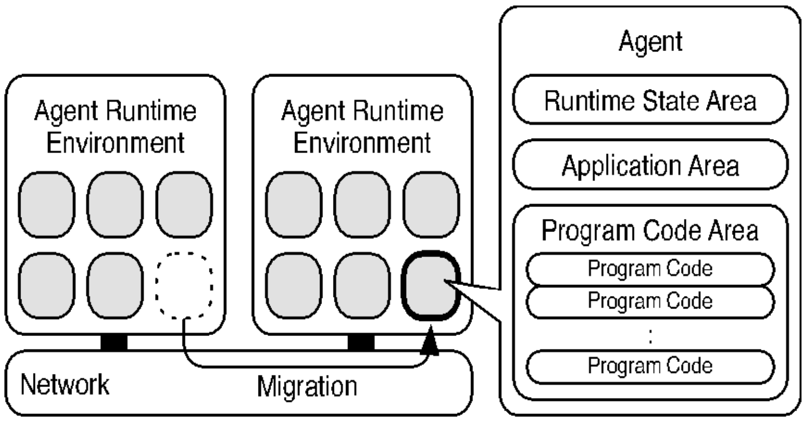


Fig. 1. An architecture of a mobile agent system.

4.2. One-to-One Communications between Mobile Agents

In a distributed system, generally, a communication protocol has types, which are *synchronous* or *asynchronous* and *pull* or *push*. Therefore, we apply these two types of protocols into the communication protocol as messaging between mobile agents. Fig. 2 shows one-to-one communication protocols between mobile agents.

4.2.1. Push with asynchronous

This communication method (Fig. 2a) sends a message from a source agent to a destination agent in only one direction. This method is used when the reliability assurance that the message has arrived at the destination is unnecessary. This nature is similar to the User Datagram Protocol (UDP) [14] in the transport layer of the Internet protocol suite. In a distributed system, particularly a microservice architecture, a communication protocol in the application layer of the Internet protocol suite is used for cooperation between microservices. There are, also, cases where multi-hop is used between microservices. Therefore, it is necessary to realize the properties of the communication protocol like UDP in the application layer as the communication protocol of mobile agents.

4.2.2. Push with synchronous

This communication method (Fig. 2b) sends a message from a source agent to a destination agent in only one direction. This method is used when the reliability assurance that the message has arrived at the destination is necessary. This nature is similar to the Transmission Control Protocol (TCP) [15]. When It is required that the source agent knows the message has arrived at the destination agent and the message passes through a multiple microservices, this is realized at the application layer as the communication between mobile agents.

4.2.3. Pull with asynchronous

This communication method, Fig. 2c, sends a request message from a source agent to destination agent, and source agent does not wait for the return of the response message from the destination agent. This communication method can implement with combining *Push with Asynchronous* (Fig. 2a). Because this communication method can be considered to be frequently used in a distributed system and the microservice architecture, we define as a basic communication method for mobile agents. Such a nature is useful, for example, when implementation of the eventual consistency [16] in the data coherence model as providing BASE (basically available, soft state, eventual consistency) semantics [17]. Since coordination of data consistency in a distributed system is an essential technique for increasing the throughput of a distributed system. Since our proposed architecture is trying to facilitate division and integration of microservices, this communication method is important to prevent a decrease in performance.

4.2.4. Pull with synchronous

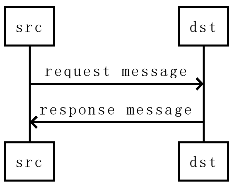


Fig. 2a. Push with Asynchronous.

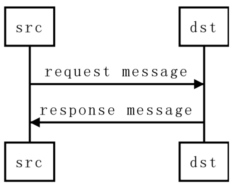


Fig. 2b. Push with Synchronous.

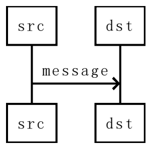


Fig. 2c. Pull with Asynchronous.

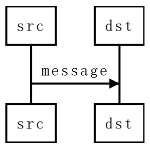


Fig. 2d. Pull with Synchronous.

Fig 2. One-to-one communications between mobile agents.

This communication method (Fig. 2d) sends a request message from a source agent to a destination agent, and source agent waits for the return of a response message from the destination agent. This communication method can be implemented with a combination of *Push with Synchronous* (Fig. 2b). In a distributed system, This communication method is necessary to implement distributed transactions and distributed consensus such as Paxos [18].

4.3. Many-to-Many Communications among Mobile Agents

In distributed systems, distributed transactions and distributed consensuses, such as two/three-phase commit protocol, Paxos [18], etc., are an important mechanism. Our proposed architecture is intended to implement the distributed transaction and the distributed agreement with a unified application layer communication protocol called message exchange between mobile agents and agents. It is easy to imagine that this is lower performance than existing high-performance products. However, by using a unified communication protocol throughout the system, there is the flexibility to change various requirements and the advantage that it is possible to understand a flat system independent of technical constraints and to coexist domains. Our proposed architecture adopts the latter over the current performance.

5. Design for Framework

5.1. Dividing and Merging for Processes

Fig. 3 shows a conceptual diagram of process dividing and merging. A programmer can migrate any process to any node. When a process is migrated, a local communication and a remote communication is handled transparently via the system. In general, a communication delay is greater in remote communication than in local communication. Thus, there is a tradeoff between redundancy due to system decentralization and communication delay by remote communication. In changing the construction of the system, it is important to balance the quality of the domain and the quality of the performance. In this our architecture, even if the process is migrated to an arbitrary node, inter-process communication is maintained whether it is local or remote. This makes it easier for programmers to find boundaries that can balance process dispersion and performance without being aware of communication types. This property is important in the domain-driven design, it makes easy to find for a better domain, and the changeability of the system can be improved.

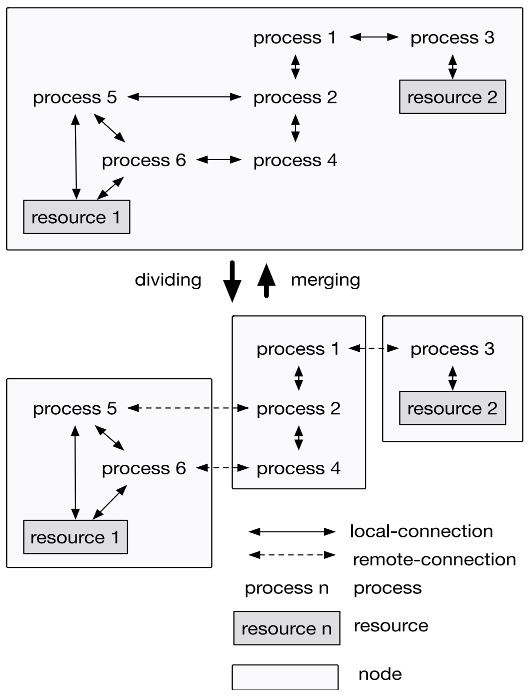


Fig. 3. A conceptual diagram of distributed dividing and merging of processes.

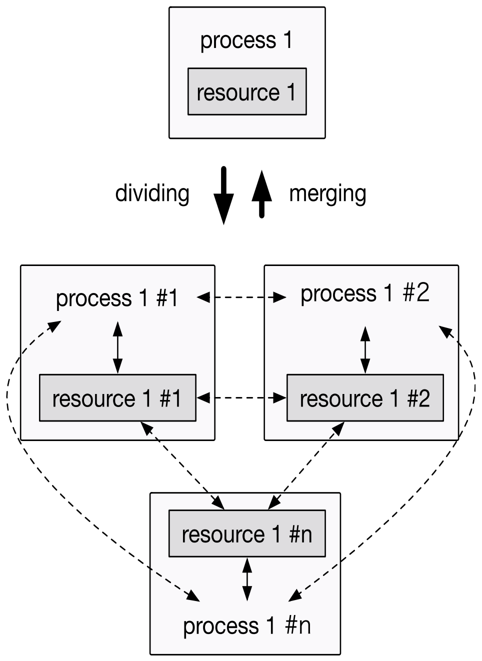


Fig. 4. A conceptual diagram of distributed dividing and merging of data (resource).

5.2. Dividing and Merging for Data

Fig. 4 shows a conceptual diagram of resource dividing and merging. A programmer can clone arbitrary resources by duplicating or dividing it to an arbitrary number. Our framework is responsible for ensuring the connection with the divided resources and an ability of look-up divided resources. Whether the divided resources take the master-slave model, multi-master model, Paxos [18] for consensus protocol or the like is realized by one layer above the framework. If a programmer just want to physically divide the resources, you can do so by dividing and merging of the process described above section.

In this our approach, by realizing the process distribution and the data distribution with different aspect, the programmer can understand separately the domain design of the program and the guarantee of consistency to the data.

6. Related Works

6.1. Mobile Agents and Microservices

Provalets [19] proposes a method to manage microservices using mobile agent technology. Fluid [20] proposes a transportable and adaptive web service model like a mobile agent architecture. [21] proposes a Performance-based cost models for improving web service efficiency through dynamic relocation.

However, these are not has been made for processes to extract microservices from a monolithic service and reconstructing a monolithic service to microservices.

Our proposed approach is extracting microservices from a monolithic service and reconstructing a monolithic service to microservices.

6.2. Web Services Composition and Microservices

Automatic web service composition like a mobile agent architecture, such as [22], is an active research area [23]-[25]. Self-organization and self-management of processes including microservices too. In the [26], associations of multiple microservices are explained by using a tree graph of services' types and graph trees of services' instances, and proposals are made to describe the concept of service orchestration and load balancing for automatic management of microservices that this work reuses much of the ideas from [27] described them. These approaches aim to composite existing services.

[28] is intended to dynamically develop and reconfigure services based on the user's request during execution. It is similar to the idea of microservice to create small reusable services based on user's request or to combine them as a single service. However, in general, it is not clear how much demand can be expected at the stage of launching services. It may be more efficient to develop as a monolithic service than to create many small dynamic software components that can be dynamically reused like microservice from the beginning. The problem we are focusing on is whether we can provide a means to easily migrate from monolithic service to microservice.

6.3. Cloud Computing

In the area of cloud computing, researchers are underway to adjust the physical location of virtual machines (VMs) for the quality of service (QoS) control. [29] surveys QoS in cloud computing. Discussions related to mobile agents and microservices in the research area of cloud computing are focused on the migration of VMs such as [30]-[32]. In the live migration of a VM, migration is performed with the interfaces and various resources in VM being fragmented and active at the same time on a plurality of computers at the same time. This technology is probably considered to be useful as an idea for transparently dividing a monolithic service into multiple microservices and autonomous horizontal scaling. However, these live migration techniques for MVs do not assume that a VM will be divided into two or more instances during live migration.

6.4. Distributed Database

Ref. [33] provides efficient partitioning and allocation of data for web service compositions. This approach that partitions and allocates small units of data, called micro-partitions, to multiple database nodes, and improves data access efficiency over the standard partitioning of data. However, this approach is not premised on consistency with units of microservices based on business requirements which are important in microservice. However, it depends on the three-tier architecture, and the proposal of this paper, an extraction of microservices from a monolithic service, has not been proposed.

Azure Cosmos DB [34] is a globally distributed database that can choose a consistency level from strong, bounded staleness, session, consistent prefix, or eventual [35] and choose a database model from collections, graphs, or tables [36].

On the other hand, our proposal is to easily divide monolithic web services into microservices while considering consistency with division units based on business requirements. In this case, in order to localize the scope of influence by service maintenance, it is the goal to dynamically lower the degree of coupling between services and to divide the service while increasing the degree of condensation of the service.

7. Conclusions

This paper has discussed an application of mobile agent technology to microservice architecture and shows requirements for designing a mobile agent framework to manage microservices on the web.

This discussion shows the following three propositions.

- Data Distribution: We propose to extract a data model with a strong association strength between models as a microservice while measuring the frequency of transactions of data generated by multiple data models and the communications traffic of data. A mobile agent is a useful aspect in cases where autonomy and transfer between computers are required.
- Process Distribution: A microservice having the function of processing data has the greatest influence on the network distance between the microservice having the data to be processed, the microservice as the processing result output object, and the end user's computers. It is required to migrate to a microservice with low cost. Such a requirement is a field that the mobile agent is good at, and it seems that affinity between autonomous processing dispersion and mobile agent is high.
- Domain-Driven Design: By adopting a mobile agent, which is a small software component capable of both data retention and data processing, as a system platform, it is possible to realize a junction capable of achieving both loose couplings in the problem domain and high condensability. It is thought that it can be flexible and easily extracted as a microservice.

In future work, we design and develop the framework including programming language and its runtime platform for reconstructing microservices from a monolithic service.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 15K15982.

References

[1] Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). *Microservice Architecture: Aligning Principles, Practices, and Culture.* California: O'Reilly Media, Inc.

[2] Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems.* California: O'Reilly Media, Inc.

[3] Lewis, J., & Fowler, M. (2014). Microservices: A definition of this new architectural term. Retrieved from January 5, 2018, from http://martinfowler.com/articles/microservices.html

[4] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-Based Software Architectures*. Unpublished Ph.D dissertation, University of California, Irvine, CA, USA.

[5] Hansson, D. H. (2018). *Ruby on Rails.* Retrieved January 5, 2018, from http://rubyonrails.org/

[6] Foundation, C. S. (2018). CakePHP cookbook documentation. *Cake Software Foundation.*

[7] Edlich, S. (2018). *NoSQL Databases.* Retrieved January 5, 2018, from http://nosql- database.org/

[8] Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Boston: Addison-Wesley Professional.

[9] Belshe, M., Peon, R., & Thomson, E. M. (2015). *RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2).* Retrieved January 5, 2018, from https://tools.ietf.org/html/rfc7540

[10] Fuggetta, A., Picco, G. P., & Vigna, G. (1998). Understanding code mobility. *IEEE Transactions on Software Engineering, 24(5),* 342-361.

[11] Object management group, Inc. (1997). *Mobile Agent System Interoperability Facilities Specification.*

[12] Foundation for intelligent physical agents. (2004). *FIPA Agent Management Specification (SC00023K).*

[13] Foundation for intelligent physical agents. (2002). *FIPA Abstract Architecture Specification (SC00001L).*

[14] Postel, J. (1980). *RFC 768: User Datagram Protocol.* Retrieved January 5, 2018, from https://tools.ietf.org/html/rfc768

[15] Cerf, V., Dalal, Y., & Sunshine, C. (1980). *RFC 675: Specification of Internet Transmission Control Program.* Retrieved January 5, 2018, from https://tools.ietf.org/html/rfc675

[16] Vogels, W. (2009). Eventually consistent. *Communications of the ACM*, *5(1),* 40-44.

[17] Pritchett, D. (2008). Base: An acid alternative. *ACM Queue*, *6(3),* 48-55.

[18] Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems*, *16(2),* 133-169.

[19] Paschke, A. (2016). Provalets: Component-based mobile agents as microservices for rule-based data access, processing and analytics. *Business & Information Systems Engineering*, *58(5),* 329-340.

[20] Pratistha, I. M. D., & Zaslavsky, A. (2004). Fluid: Supporting a transportable and adaptive web service. *Proceedings of the 2004 ACM Symposium on Applied Computing* (1600-1606).

[21] Pratistha, D., Zaslavsky, A., Cuce, S., & Dick, M. (2005). Performance based cost models for improving web service efficiency through dynamic relocation. *Proceedings of the 6th International Conference on E- Commerce and Web Technologies* (pp. 248-257).

[22] Wang, P., Ding, Z., Jiang, C., Zhou, M., & Zheng, Y. (2016). Automatic web service composition based on uncertainty execution effects. *IEEE Transactions on Services Computing*, *9(4),* 551-565.

[23] Immonen, A., & Pakkala, D. (2014). A survey of methods and approaches for reliable dynamic service compositions. *Service Oriented Computing and Applications*, *8(2),* 129-158.

[24] Dustdar, S., & Schreiner, W. (2005). A survey on web services composition. *International Journal of Web and Grid Services*, *1(1),* 1-30.

[25] Garriga, M. (2017). Towards a taxonomy of microservices architectures. *Proceedings of the 1st Workshop on Formal Approaches for Advanced Computing Systems, located at the 15th International Conference on Software Engineering and Formal Methods* (pp. 1-15).

[26] Toffetti, G., Brunner, S., Blöchlinger, M., Dudouet, F., & Edmonds, A. (2015). An architecture for self-managing microservices. *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud* (pp. 19-24).

[27] Karagiannis, G., Jamakovic, A., Edmonds, A., Parada, C., Metsch, T., Pichon, D., Corici, M., Ruffino, S., Gomes, A., Crosta, P. S., & Bohnert, T. M. (2014). Mobile cloud networking: Virtualisation of cellular networks. *Proceedings of the 21st International Conference on Telecommunications* (pp. 410-415).

[28] Silva, E., Ferreira Pires, L., & Van Sinderen, M. (2009). Supporting dynamic service composition at runtime based on end-user requirements. *Proceedings of the International Workshop on User-Generated Services, Located at the 7th International Conference on Service Oriented Computing* (pp. 1-10).

[29] Ardagna, D., Casale, G., Ciavotta, M., Pèrez, J. F., & Wang, W. (2014). Quality-of-service in cloud computing: Modeling techniques and their applications. *Journal of Internet Services and Applications, 5(1),* 1-17.

[30] Wei, B., Lin, C., & Kong, X. (2011). Dependability modeling and analysis for the virtual data center of cloud computing. *Proceedings of the 2011 IEEE International Conference on High Performance Computing and Communications* (pp. 784-789).

[31] Anandkumar, A., Bisdikian, C., & Agrawal, D. (2008). Tracking in a spaghetti bowl: Monitoring transactions using footprints. *ACM SIGMETRICS Performance Evaluation Review*, *36(1),* 133-144.

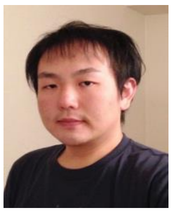
[32] Melo, M., Maciel, P., Araujo, J., Matos, R., & Arajo, C. (2013). Availability study on cloud computing environments: Live migration as a rejuvenation mechanism. *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (pp. 1-6).

[33] Kish, A. V. (2016). *Efficient Partitioning and Allocation of Data for Workflow Compositions*. Unpublished Ph.D dissertation, University of South Carolina, South Carolina.

[34] Microsoft Corporation. (2018). *Azure Cosmos DB – Globally Distributed Database Service.* Retrieved from January 5, 2018, from https://azure.microsoft.com/en-us/services/cosmos-db/

[35] Microsoft corporation. (2017) *Tunable Data Consistency Levels in Azure Cosmos DB.* Retrieved January 5, 2018, from https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels

[36] Microsoft corporation. (2018) *How to Partition and Scale in Azure Cosmos DB.* Retrieved January 5, 2018, from https://docs.microsoft.com/en-us/azure/cosmos-db/partition-data



Masayuki Higashino was born in 1983. He received his B.Eng., M.Eng., and D.Eng. degree from Tottori University, Japan, in 2008, 2010, and 2013, respectively. He has been an assistant professor at Tottori University since 2015. His research interest includes mobile agent systems, distributed systems, and network architecture. He is a member of the IPSJ, JSSST, IEEE CS/ComSoc and ACM.

Toshiya Kawato was born in 1989. He has been a graduate student of Tottori University for his doctoral degree. He is graduated from National Institute of Technology, Yonago College and received his B.Eng. degree from National Institution for academic degrees and University Evaluation in 2012. He has been a technical staff at Tottori University since 2012. His interest includes distributed storages. He is a student member of IPSJ.

Takao Kawamura was born in 1965. He obtained his B.Eng., M.Eng. and Ph.D. degrees in computer engineering from Kobe University, Japan in 1988, 1990 and 2002, respectively. Since 1994 he had been in Tottori University as a research associate and has been in the same university as a professor in the Faculty of Engineering since 2009. His current research interests include mobile-agent systems and distributed systems.