Migrating Web Applications from Monolithic Structure to Microservices Architecture

Zhongshan Ren, Wei Wang*, Guoquan Wu, Chushu Gao, Wei Chen, Jun Wei, Tao Huang
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Science, Beijing 100190, China
University of Chinese Academy of Sciences, Beijing 100049, China
{renzhongshan13, wangwei, gqwu, gaochushu, chenwei, wj, tao}@otcaix.iscas.ac.cn

ABSTRACT

In the traditional software development and deployment, the centralized monolithic is always adopted, as the modules are tightly coupled, which caused many inconvenience in software DevOps. The modules with bottlenecks in monolithic application cannot be extend separately as the application is an integral part, and different module cannot use different technology stack. To prolong the lifecycle of the monolithic applications, its need to migrated it to microservice architecture. Due to the complex logic and large number of third party framework libraries depended, get an accurate comprehensive of the application characteristics is challenging. The existing research mostly based on the static characteristics, lack of consideration of the runtime dynamic characteristics, and the completeness and accuracy of the static analysis is inadequate. To resolve above problems, we combined static and dynamic analysis to get static structure and runtime behavior characteristics of monolithic application. We employed the coupling among functions to evaluate the degree of dependence, and through function clustering to achieve the migration of legacy monolithic applications and its data to microservices architecture. Through the empirical study of migrate the typical legacy project to microservices, it is proved that we proposed method can offer precise guidance and assistance in the migration procedure. Experiments show that the method has high accuracy and low performance cost.

CCS CONCEPTS

• Software and its engineering→ Software evolution; Maintaining software; E-commerce infrastructure; • Information systems → RESTful web services; Service discovery and interfaces;

KEYWORDS

monolithic application, microservices, application migration, function clustering

ACM Reference format:

Zhongshan Ren,Wei Wang,Guoquan Wu,Chushu Gao,Wei Chen,Jun Wei and Tao Huang. 2018. Migrating Web Applications from Monolithic to Microservices Architecture. In Proceedings of *Internetware'18, Beijing, China, September 16, 2018, 10 pages.* https://doi.org/10.1145/3275219.3275230

Internetware '18, September 16, 2018, Beijing, China

ACM ISBN 978-1-4503-6590-1/18/09…$15.00
https://doi.org/10.1145/3275219.3275230

1 INTRODUCTION

In the traditional application, all the business components are packaged together, distributed and deployed as a whole, this development and deploy pattern is called monolithic [1]. To realize the highly available and flexibly, the monolithic application deployed as a whole, and the load balancing carrying out at the front end using a load balancer. When the application meets performance bottleneck, it is usually has some limitation in one of the modules, rather than the overall of the system components [3]. When it comes to the monolithic architecture system, we cannot make replicated the relevant component or module for extending, and deploying the entire application across multiple nodes will cause waste of resources.

The microservice architecture is a lightweight, miniaturized development and operation mode. Each microservice deployed independently and follow the single responsibility principle [4], different microservice can use different technology stack, and update independently [5]. Microservices use lightweight HTTP protocol for communication and data exchange. For applications, which need to realize the high-concurrency and high-capacity system, microservices technology architecture is a good choice. Under the microservices architecture, the microservice contains the bottleneck components can deployed by multiple copies to resolve the performance and scalable problems. Microservice technology architecture provides an effective solution for solving the problems such as large project team, complex and time-consuming software update iteration, and difficult maintenance in the continuous integration and release of large application systems [7]. At present, many internet companies carried out the migration from monolithic to microservice architecture, such as Amazon, Netflix, Google, IBM, Uber, Alibaba etc. al.

Migrating the monolithic application to microservices is transform the monolithic legacy application to the microservice architecture following the microservice principles [9]. Based on the static characteristics of legacy systems, the existing work combines search and clustering methods to realize the extraction of microservices from legacy applications. Then based on the preceding results, using microservices technology to reconstruct the legacy monolithic applications. The disadvantage of existing works is that static analysis cannot obtain the application runtime behaviors characteristics that can only be determined when it running, for example, uncertainty due to polymorphism and the behaviors determined by configuration files and user input parameters. In microservice migration, due to the incompleteness

of the static structural characteristics of the application and the lack of invoke frequency information, the application's behavior characteristics cannot be full described. Therefore, the microservice division using only the static structural characteristics of the application can easily lead to inappropriate microservice partition.

In this paper, we proposed a program analysis based approach to migrate the legacy monolithic application to microservice architecture. We get the class inheritance and dependency relationship and the invocation among functions in legacy applications based on the static analysis of the application's source code or binary code. After that, we use the dynamic analysis approach to obtain the runtime features of it, such as function invokes between the relations and interactions between classes, and use it as a static characteristic data supplement, to improve the integrity and accuracy of static model. The contributions of this paper include:

(1) We proposed a model that combines static analysis and dynamic analysis for migrating the legacy web applications to microservices technology architecture based on application behavior feature.

(2) Based on the application behavior model, this paper proposes a microservice division method based on clustering of legacy application functional correlations by using class and function association analysis and application domain model and data flow analysis.

(3) Through the empirical research on the scalability and performance differences of different partitioning results, the key factors affecting the migration of legacy systems to microservice architecture are determined, and the performance of the system is improved and the scalability is enhanced and many other expected optimization effects.

2 MOTIVATION

Microservices architecture had proposed to deal with continuous integration and continuous delivery in the large application system development and operational scenarios, the key problem to solve is the problem for application updates and iteration. In the development of a new application, microservices design principle and patterns can used in directly, but in the case of the existing business critical systems, the existing legacy systems need to transform and migrate to microservices technology architecture to adopt new technology and improve the development efficiency.

To solve the challenges and problems faced by legacy monolithic application, such as inadequate of service capacity, limited extension ability, less scalable and fault-tolerant ability, microservices technology architecture had proposed by the researchers. With the progress in software developing technology, software system architecture has been gradually turn to distributed system and cloud platform, container and related technology becoming mature, speed up this procedure. As the interaction among microservices always across hosts or containers, the communication time consuming over the network is always the main concerns of the migration procedures. The monolith application commonly in high functional cohesion, interact with each other in the same process, the interaction between the microservices need to cross the host node or container, resulting in a number of uncertainties, such as transmission delay, network fault, repeatedly submit, state detection, failover, and so on.

As the problems mentioned above exists, in the industry scenarios the application of microservices architecture applied in real-time trading system, critical-mission business system and transaction intensive application system very rare. Most of the migration is about simple and small systems, and the solution to the challenges and problems of large business systems migration are very limited.

3 METHODOLOGY

In the migration of the legacy monolith system to the microservices architecture, the structural characteristics of the existing applications need analyzed. Firstly, we obtained the static and dynamic characteristics of application through static analysis of the source code and tracing data during application's execution. Then, based on the architecture characteristics of microservices, we get the collection of candidate microservices through feature clustering. Then, we adopts the development technology related to microservices to transform the monolith application system to the microservices technology architecture. Finally, we evaluated the candidate microservices implementations to obtain the best service migration strategy.

3.1 Approach Overview

In the migration of the legacy system from monolithic to the microservices architecture (MSA), as shown in Figure 1, we analyzed the behavior characteristic and modeling it at the first step. We use static analysis of the application source code to get the static trait of the monolithic system, and obtained the runtime dynamic features through the dynamic tracing. The full and comprehensive structure and functional interaction characteristics of the legacy applications generated by combining the static and dynamic characteristics. Based on domain knowledge about the monolith system's structure, function and features obtained in the first step, we get the collection of candidate microservices by feature clustering of the legacy applications. The final microservices obtained based on evaluation, analysis of the candidate microservices sets through the design principles of single responsibility, highly cohesive, loosely coupled, and the interaction between the microservices should be at a low level. Finally, the microservices system constructed from the selected candidate microservices with Spring Boot and other related microservices technologies.

3.2 Legacy Application Analysis and Modeling

Through the static analysis of the application source code, we get the class hierarchy architecture (CHA) and invoking relationship between functions, and build the dependency graph of the application modules, classes and functions. The static component relationship of the monolith application constructed by applying

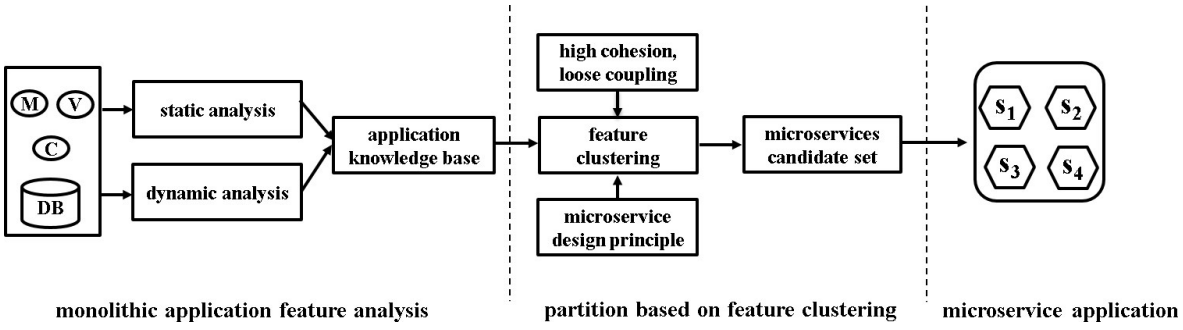


Figure 1: Approach overview

static analysis results in source level. In the process of migration the legacy system to microservices architecture, it is necessary to use the coupling degree between the functional modules and the frequency of the call as a reference.

Through static analysis method for application of the overall relationship between the structure and module, we get the invocation chain among the functions; the relationship reflects how the application functions linked closely. In the process of migrate the legacy application to the microservices architecture, rules that needs to be follow: (1) Make sure that the function with more closely linked were divided to the same service, to reduce time costs induced by the communication between the microservices. (2) High cohesion and loose coupling, the basic principle of software development should be adhere.

How to divide the legacy application or in another word extract the candidate microservices is the key point to the migration process. The static analysis of the legacy application provide the structure information and the potential relationship of the classes, methods or even variable members. We use the features that we can obtained from source codes, design documents, and developer information to improve the application partitioning.

In the process of migrating legacy systems to a microservices architecture, the partition of legacy applications based on the degree of coupling between application functional modules and the degree of invocation frequency. Through to the application source code static analysis for the application of class hierarchy, dispatching relationship between function and build the application function modules, classes, functions, a dependency graph, which structure characteristics of the application. Through static analysis method for application of the overall relationship between the structure and module, can get the function between the invocation chain, call the relationship reflects the application features and functions of closely linked. During legacy systems to the service architecture in the process of migration, through divide the function of closely linked to the same service, help to reduce due to the communication between the microservices time costs, also accord with the software development of the basic principles of high cohesion and loose coupling. Instrumentation based on the application in the way of tracing the implementation process of application, application in the process of execution were obtained through the dynamic tracing method call features and user behavioral characteristics such as the use of information in application system. Dynamic analysis for the static analysis of the uncertain real executed unable to count objects, application of the problem of user access frequency and static analysis can make up for the dynamic tracing the recall ratio and the coverage problem, so using a combination of static analysis and dynamic analysis, to achieve a comprehensive description of application characteristics.

In the analysis on the legacy application, we adopt the static interaction model, runtime dynamic interaction model, structure model, the developer's contribution model, to describe the structure of the application characteristics, provide reference for the application of microservices division. These different models and abstractions described as below.

3.2.1 Static Interaction Model. The call diagram composed of class and function call relations in the application can reflect the characteristics of module interaction in the application. We use static analysis to get the chain of function execution calls of the application. As shown in figure 2, the microservices candidate set extracted by searching the graph and clipping the graph. Function is obtained by the interaction relationship between functions, the invocation of the map, then through correlation evaluation index and domain knowledge will be closely linked, and conforms to the call both domain knowledge together; A candidate set of microservices through continuous aggregation. In this article, through the Abstract Syntax Tree to describe the application of static structural characteristics, we will be the legacy monolith, the feature information of the application to use its abstract syntax tree (AST_M), semantic feature $Syntax_M$ and static call graph $(CallGraph_M)$.

$$M_{Architecture} = AST_M + Syntax_M + CallGraph_M \quad (1)$$

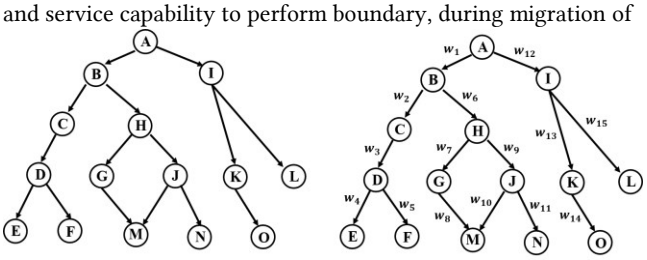
For the application of the function and method calls, use C < method, invoke, oriented > tag, use function of quantities as a reasonable division, Interactive < Indegree, Outdegree > out of and into the degrees, and the invocation of the relationship between function correlation of building applications. We use the coupling relation of function to analyze the output and input of function, and according to some prior knowledge, we can apply data access layer and data storage layer.

The application's class and function interaction relations, we use $R = G(V,E) \cup I(i_1, i_2 \ldots, i_n)$ to express it. In which, R is the interaction relationship between classes can be get according to the application's structure and the relationship between each other by static analysis, the G(V,E) is the call graph described with vertexes and edges, the vertex as the function and edge is the invoke relation, and I for as the coupling degrees. The static feature uses the calling relation, the frequency and other information appearing in the class to represent the degree of dependence. The coupling relationship between classes described through the hierarchy of classes and the calling characteristics of each other. Another is to evaluate the possibility of class migration to microservices based on the characteristics of microservices through the actual requirements when building microservices.

The method set (M, C) used to construct the graphical representation of the software application system and refine the static analysis model applicable to the application of microservices extraction. The function calls and functions of the application are coupled, the domain knowledge is analyzed and sorted out, and the migration model of the application microservices is constructed under the microservices scenario.

3.2.2 Application Dynamic Model. As the application information obtained by the static analysis lacks the characteristics of the application run time, the interaction frequency among functions cannot be collect, which caused the application characteristics model not so precise. By means of dynamic tracing, the application runtime information recorded and the dynamic behavior characteristics of users can be obtain. However, due to the low code coverage of the dynamic analysis method, unexecuted program path will not recorded. Dynamic analysis have the risk that cannot cover all users input, so the feature of the application, which got was incomplete. In order to reproduce the features of the application as fully as possible, we supplement static analysis with dynamic analysis results. Through tracing the users' use of legacy system, the user's usage characteristics during application execution generated. Since the behaviors of users remain unchanged in the application systems of different architectures, the user usage characteristics captured by dynamic analysis as the basis for constructing the application of microservices architecture. By collecting the information, we add the call relationship between functions that are not available in static analysis. The dynamic scheduling frequency combined with the dependence obtained from static analysis to form the interactive characteristics of the application. In addition, the accumulation of legacy applications in the process of running a large number of runtime log information and trace data, we can get the invoke relation through the analysis of the log information. The system's log and error information can also helpful to obtain the interactive relationship characteristics of the application. We adopts the customer behavior model graph (CBMG) to represent the components interaction of the application and use the statistical data to calculate the transition probability. The class and function invoke relations can be generated from the dynamic tracing data, the call graph obtained by static analysis and dynamic analysis shown in Figure 2, the dynamic method can infers the coupling degree of the class and functions from the interaction frequency.

3.2.3 Constraints of Microservices Construction. The determination of microservices boundaries is the core of the migration, during the microservices extraction need to combine the service and service capability to perform boundary, during migration of



(a) call graph obtained by static analysis (b) dynamic tracing with the frequency

Figure 2: Application Component Interactions Represent by Calling Trace

legacy systems to the service architecture, we define the boundary of the microservices from legacy systems to determine the functions of the service. The process of constructing a microservice needs to follow the characteristics of a microservice, as high cohesion and loose coupling. The evaluation principles include:(1) reduce the interaction frequency of microservices based on the evaluation of coupling degree; (2) reduce message communication overhead in microservices; (3) the deployment and choreography of microservices should be as low as possible.

After constructing the candidate set of microservices, we use the microservice design principles to evaluate the quality of the candidate microservices. Considering a situation that it has a number of different partitions, adopting what standard to building the microservices. The similarities and differences between different divisions evaluate the. Deployment testing and capability testing of microservices carried out. The microservices candidate set examined will be the final candidate microservices set, and it is up to the developer to select the appropriate solution implementation.

In order to evaluate the connection and jump probability between class and method structures in the application, we use transition probability to represent the jump probability from node a to node b $P(b|a)$ using markov chain model to represent migration characteristics between nodes or functions in the application. Through the static analysis method for application of the basic relations of the scheduling and static scheduling frequency, again using dynamic tracing method for application of transfer between the various functions of the execution path and frequency, combined with the data of both to get a complete application function of interaction matrix.

Model building: application interaction models, feature models, DSM-derived classes, functions, and their invocation relationships. In microservices selection, the follow rules should be consider: (1) stateless components, (2) stateful components; (3) transaction consistency; Transactional consistency assurance in RESTful mode, the tradeoff between consistency assurance and service performance. (4) message-driven, event-driven architecture; (5) microservices interaction and process flow choreography, and the pattern microservices interaction and organization.

Microservices evaluation and measurement: the number of interaction interfaces, the cohesion of the structure, the ease of deployment, the number of classes, the number of lines of code. Statistical analysis evaluation criteria: the application of the coupling between modules, can determine the close degree of connection between functional modules. The application's function module with correlation, due to its frequent interaction, so if split in different microservices, will lead to two modules interact to produce a large number of communication and accompanied by a network transmission delay, thus resulting in the whole system performance degradation. Therefore, in the partition of micro-services, the functional modules with high coupling should be migrated to the same microservices.

During legacy systems to the service architecture in the process of migration, through divide the function of closely linked to the same service, help to reduce due to the communication between the microservices time costs, also accord with the software development of the basic principles of high cohesion and loose coupling. Therefore, we consider the degree of coupling between application functional modules and the degree of invocation frequency as a standard for legacy application partitioning

3.2.4 Migration Rules. In order to implement a generic semi-automated migration approach that requires less developer involvement, an application evaluation model for migration to microservices needs to be built. So as to provide reference and guidance for the migration of microservices. Through empirical research, we summarize a series of migration patterns. We extract microservices from the monolith application based on clustering according to the value of the coupling evaluation function. In the field of specific classification in this paper, we based on existing knowledge to pretreatment of application system, the specific division and microservices work are:

(1) Based on the importance of the business logic, divided the functions to general and key functions.

(2) According to the interaction with other functional functions, there is interaction; there is no interaction or less interaction.

(3) From the perspective of invoke relation: only invoke by other functions and no call other function functions; Provide services directly by using other functions but not used by others.

(4) Whether there is a circular dependency: linear call chain, loop call chain, self-call or loop iteration.

(5) When the division or boundary is determined, there is a definite division function and a deterministic function. If there is more than one choice, as it can placed in different microservices function called swing function, in view of the swing function to set the degree of freedom, the degree of freedom for the optimization of the function can be used as a high factor.

The construction standard of microservices and the principle of partitions can constitute the functional function characteristics of microservices.

3.3 Application Characteristics Clustering

How to determine microservices boundary is the key to the micro migration of service, this article through to the structure of the application features, behavior modeling, and based on this model to build a description of the application form of the figure, to using the clustering method, realizing the function of application of the aggregation. According to the interaction characteristics and domain knowledge between application function modules, divided it into different microservices. On the concrete implementation, this paper adopts conditions of clustering method, the evaluation function in the use of frequent item clustering using the clustering process of constraint function of adjustment to adjust and control. The microservice extract procedure shown in Figure 3.

In the legacy system represented by call relation graph, the dependency of interactive node represented by vector matrix. When clustering, the distance from the center point taken as the measure when clustering is executing. In this paper, the degree of coupling in function call relation expressed by cosine distance, such as formula (2). By using the k-means hierarchical clustering method, we will cluster the behavior characteristics and calling relations of the application. Here, we first determine the core business of the application according to the interactive behavior characteristics of the application. Clustering and evaluation carried out according to the distance from the center point.

$$\cos\theta = \frac{x_1x_2 + y_1y_2}{\sqrt{x_1^2+y_1^2}\sqrt{x_2^2+y_2^2}} \quad (2)$$

The degree of correlation between two functions expressed by the correlation coefficient as formula (3).

$$\rho_{xy} = \frac{Cov(X,Y)}{\sqrt{D(x)}\sqrt{D(Y)}} = \frac{E((X-EX)(Y-EY))}{\sqrt{D(X)}\sqrt{D(Y)}} \quad (3)$$

In the process of clustering, if there is A variety of different classification results, we use Jaccard similarity coefficient to compute the similarity of two different partition, if you have A, B two microservices division, we use Jaccard similarity coefficient to measure the similarity of two. When the similarity is less than $\delta = 0.35$, it is used as a different partition; otherwise, it is considered the same partition.

$$\mathrm{J(A,B)} = \frac{|A \cap B|}{|A \cup B|} \quad (4)$$

In order to evaluate the accuracy of clustering results, this paper use SSE (the Sum of the Squared Error), shown in formula (5) as the clustering objective function, two running k-means cluster of two different, SSE a smaller is better clustering results.

$$\mathrm{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2 \quad (5)$$

In the formula (5) K represents K clustering centers, c_i represents the first few centers, and dist represents Euclidean distance, as shown in equation (6).

$$\frac{\partial}{\partial c_k} SSE = \frac{\partial}{\partial c_k} \sum_{i=1}^{K} \sum_{x \in C_i} (c_i - x)^2 = \sum_{i=1}^{K} \sum_{x \in C_i} \frac{\partial}{\partial c_i} (c_i - x)^2 \quad (6)$$

Through the clustering of monolith application's function call graph, we get the functions which have the more tightly interaction relationships within the application. The basement functional components of the legacy application will be replaced by new components build with new technology stack meet the

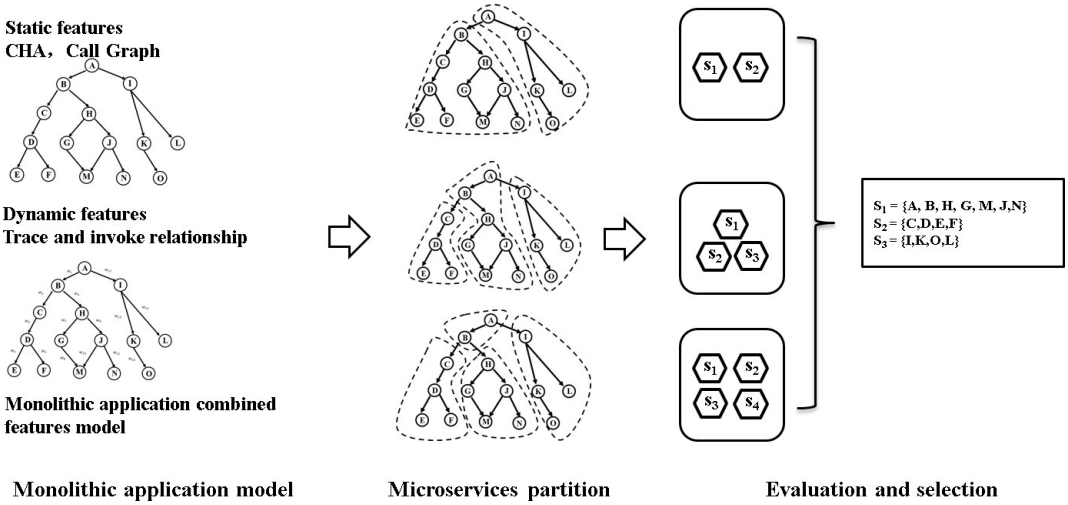


Figure 3: Determination microservices boundary based on application feature clustering

microservices standards, such as messaging, data access, interactive communication modules. First, we obtained the key functional component of the legacy system through key function identification method based on the interaction pattern exclude the frequently invoked basement function and universal function module. Second, we construct the legacy applications interactive model based on application static and dynamic analysis result, supplement with history data of user behaviors. Third, we use hierarchy clustering on the legacy monolith application interactive model for microservices candidates. As we removed the basement components and improved the legacy model through combining the static and dynamic analysis, the method is more efficient with higher accuracy.

We designed an method to migrate the monolithic legacy application to microservices architecture based on the static analysis obtained abstract syntax tree (AST), class hierarchy architecture (CHA) and call graph. Through the domain knowledge of the legacy application and program analysis, we got the core components, and we use it as the central point to clustering. For the microservices candidates set obtained by clustering, we evaluated them by the microservices design principles. The candidate microservices set which passed the evaluation is the ready for deploying procedure. As the clustering is a common used way to determine the object belong to which cluster, so we described the algorithm in natural language way as follow.

Algorithm 1: determination of microservices boundary

Input: $M_{Architecture}$: static features of monolithic application: class and function relationship and interaction features.

Dynamic Tracing data: the application runtime behavioral feature tracing data

MaxIteration: maximum number of iterations

N: the number of microservices partition into

Output: Microservices and the functional modules they comprise.

1. Based on the interaction parameters of the application system, constructing the application behavior characteristic interaction matrix.
2. Calculate the distance from the center point of each node according to the interactive characteristics of the application, and construct the adjacency matrix of nodes;
3. Let S as the microservices number and then applying K-Means clustering on function level.
4. Use domain model evaluate the candidate microservices set, and select which satisfy high cohesion and loose coupling constraint microservices set as a valid candidate. If there no candidate can meet the constraints, repeat it again until you get a satisfying one or the maximum number of iterations exceeded.
5. Output the microservices candidate that meet the design requirements.

The result of the partition is different depend on the parameters selected. An example of a migration result shown in Figure 4, the microservice number to partition is one to four. In different partitioning schemes, the same function or class may be divided into different microservice, which will brings different behavior characteristics and service capabilities of microservices under different partitions. It is necessary to evaluate the candidate microservice sets obtained by different divisions to obtain a more suitable for user requirements.

3.4 Data Migration

In the microservice technology architecture, it's suggest that the business logic function of the microservice and the data it operates deploying in the same instance, other microservices can only access the data through the interfaces it provided. When migrating the monolithic applications to microservice,

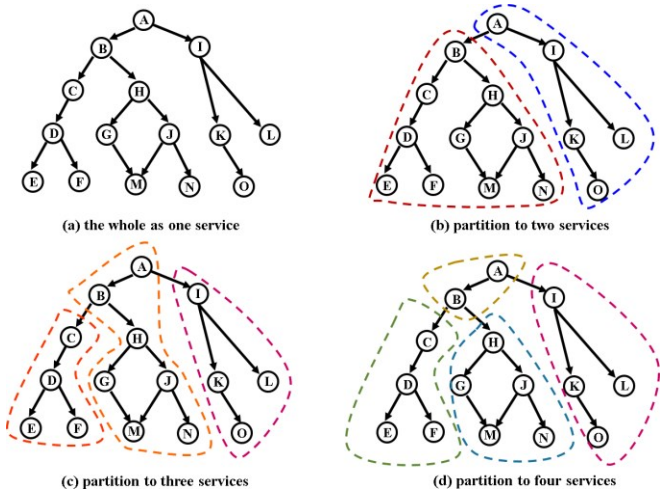


Figure 4: Different microservices partition based on method interactions

accompanied by business logic code migrating, the data needs to split into different microservices according to the dependencies of function and data and the relationship of the database tables of the monolithic application.

In microservice environment, unreasonable data partition will leads to frequent interactions among microservices, increased data transmission volume, and overall performance degradation. Conversely, reasonable data partitioning can reduce the frequency of interaction and the volume of data transmission, thus improving the performance of the entire microservices.

To partition the data of monolithic application, the pattern of the data access knowledge should be the basement of the partition procedure. The data access of the legacy monolithic application system mainly uses directly JDBC and database operation-based auxiliary libraries, such as MyBatis, Hibernate and JPA. To obtain the data access pattern of the legacy monolithic application, we use a monitoring and tracing tool called EasyAPM we developed to record the data operation and the parameter information through the instrument on the JDBC and data access class libraries, as shown in Figure 5 is the data access information of legacy DayTrader application obtained through EasyAPM.

The data storage for a legacy monolithic application dominated by a single data source, all the requests accessing the same data source. Under microservices technology architecture, it is necessary to place the function and the function operated data together, so we need according to the characteristics of the microservices for data using the original characteristics of accessing database according to the applied into different services. When partitioning the data to different microservices, it is suggest that according to the usage characteristics of the microservices and the relationship of the data to guide the partition process necessary to divide the data to multi-part.

To split the data table, we first tracing the legacy application data using characteristics, and then according to the functions of access to data entities, the application logic and its dependence data will be combined in the same service. In the microservices scenario, the business logic mainly involved in the microservices exists the use of other microservices data, the isolation and atomicity of data operations need to be guaranteed. At present, there are many ways to store operation requests by message queue or to implement data operation atomicity by distributed transaction.

The data distribution optimized according to the data synchronization requirements and data processing characteristics of access feature big data applications. The existing work focuses on dividing the access frequency of database entities according to the structure and application of database entities, which does not take into account the access requirements of application function functions to data.

In this paper, we adopts the method of trajectory tracing the implementation of the legacy system, obtained the application implementation and access to the database table history, through the use of the function of the application function of data, combined with the data table structure design norms and rules, the split data table structure, reasonable partition scheme. Get the data access characteristics of the application function, according to the data access nearby principle, in order to ensure data consistency, etc. al.

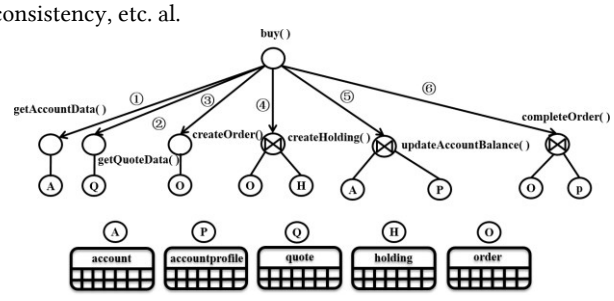


Figure 5: Legacy monolithic system data usage tracing

4 EVALUATION

In microservices environment, the developers are always worry about the frequently interaction among microservices will cause the performance decrease after the legacy monolithic applications migrated to the microservices architecture. Meanwhile most of them are sure about the microservices have many benefits in scalability, management and maintenance over the monolith architecture. As the traditional monolithic application always in large scale and with complex dependencies, the model build through static and dynamic analysis in size is very big. The volume of static analysis result presented as vertexes, edges and other related attributes is huge, which will cause the microservices extracting procedures time consuming. To evaluate our approach, we have conducted a set of experiments. All the experiments in this paper are used to address the following questions:

Q1: After migrating the legacy systems to a microservices architecture, whether microservices architecture application better at performance and scalability than monolithic ones?

Q2: What is the efficiency and performance of the analysis and clustering methods proposed in this paper? In order to prove the superiority of microservices over monolithic in service capability, performance and scalability, we design the comparison testing to verify that.

4.1 Experiment Setup

In the evaluation phase, we choose four typical performance benchmark application, DayTrader, JPetstore, TPC-W and RUBiS as system under test to compare the performance difference between microservices and monolith application. The performance comparisons between monolith and microservices applications validated by selecting different loads in nearly the same scenarios. To verify the validity and accuracy of the migration algorithm proposed in this paper, we selected 12 applications that have completed the migration from monolith to microservices in industry and open source community. The applications as IBM's microservices migration case Front Shopping, micro-company, sockshop microservices application, etc. al. were refactored to microservices architecture from monolith, we use them as the ground truth to validate the correctness of the migration solution.

In terms of test environment configuration, we used three Inspur blade server NX580. Each server configured with two Intel Xeon E5645 CPU, 16GB memory and installed Ubuntu 16.04.1 x64.

4.2 Comparison of Monolithic and Microservices

According to the method proposed in this paper, we migrated the monolithic application to microservice architecture. The basic information of the experimental objects and the results of the migration is shown in Table 1. After the migration procedure, we designed a set of experiments to compare the application's service ability and performance before and after the migration. The monolithic application is deployed on the web application server tomcat 8.5.20, and the microservices application is constructed by multiple microservices deployed independently with Spring Boot 1.5.7 and related frameworks. Under the assumed that the resources is sufficient, we gradually increasing the test load to continuously increase the load burdened by the application until the application is in error or the service capability is seriously degraded.

Table 1. Microservices research subjects and migration result

Subject	Ver.	LoC	No. of Class	No. of Microservices	No. of Interfaces
JPetstore	6.0.2	1438	24	3	16
DayTrader	3.0	19972	83	5	23
TPC-W	1.0	13912	83	5	10
RUBiS	1.4.3	12492	37	3	9

The experimental results shown in Table 2 is the service capability and performance comparison of the monolithic in left and microservice architecture in right. In low-load scenarios, the performance of the application monolithic is better than microservices architecture, the reason is that the time consumption of interaction among microservices is greater than in-process communication of the monolithic application. When the load increasing gradually, the monolithic application will reached its maximum service capacity due to the limitation of its service capability, and then a large number of out of service pages begin to appear. As the performance and capability of the microservice application under the large-scale load better than monolithic application, we can concluded that through the migration to microservices, the monolithic application's capability and the performance are all improved.

Table 2. Application performance comparison under monolithic and microservices architecture

Subject	JPetstore	DayTrader	TPC-W	RUBiS
average. throughput (req/s)	263/328	302/325	230/387	239/251
average throughput (KB/s)	5105/7123	5035/8368	4326/5218	3259/4251
maximum load	326/391	401/523	286/334	269/343
steady load	251/302	306/302	235/243	241/281
average load	265/296	300/356	229/316	238/264

The microservice application can scalable easily by the container technology, by the docker and Kubernetes etc. The result of the comparison group of the existing migration practice, we found that the method we proposed get the same result 5 out of 12 examples, and 7 out of 12 have even better performance than the existing migration.

4.3 Performance

We adopt Wala to get the static structural features of the legacy application based on static analysis. We filtered the Java basement library and common public class library without affecting the integrity of the application structure to speed up the analysis process. We use PA (package analysis), SSA (static structure analysis), CHA (class hierarchy analysis), SCGA (static call graph analysis), and CSDA (combined static and dynamic analysis) to evaluate the migration performance.

The results in Table 3 shows that the static analysis of the application can quickly obtain the result, such as package and the hierarchical structure relationship analysis, the static function call

relationship and the corresponding feature building takes a short long time. When we combining static and dynamic features to analyzing, the maximum time consuming of the whole process is 6598 seconds. The entire process can completed in up to 12 minutes, which is acceptable in the migration of legacy applications.

Table 3. Performance of legacy application feature analysis (in seconds)

Object	PA	SSA	CHA	SCGA	CSDA
JPetstore	3	150	210	1568	2689
DayTrader	9	200	291	2635	6598
TPC-W	5	320	110	985	1205
RUBiS	10	190	98	698	968

5 RELATED WORKS

As the software requirements changing and technology updating, it need to migrate from the one technology architecture to a more advanced one. The application refactoring and migration process includes technical architecture upgrades, code updates, deployment environment migration, and interaction mode changes [12]. In terms of application refactoring and migration, many researchers have studied the requirements of application migration and specific operational steps. Many researchers using similarity analysis on legacy code to reconstructs the legacy systems to multiple services and transform them to service-oriented architectures by package legacy applications into services [13, 14].

After the birth of cloud computing technology, due to the high performance, high scalability, high cost performance and flexible scalability of cloud computing technology, the migration of traditional applications to the cloud environment can achieve higher performance acceleration ratio and better software features [16]. In order to make full use of the advantages of cloud computing, academia and industry have carried out a large number of research work to transfer traditional applications to the cloud environment [8]. The related research includes refactoring the application to cloud services or migrate some properly function and data to cloud environment [17].

In recent years, with the development of computer technology, monolithic application have begun to face various bottlenecks and challenges. In order to solve the challenges faced by monolithic applications, microservice architecture was put forward [15, 19, 20, 25]. Due to the excellent characteristics of microservices, such as elastic scalability, automatic failover, load balancing, ease of deployment and distribution [21, 27, 28, 29, 31], microservices have attracted extensive attention from academia and industry. With the rise of container technology, a large number of application development models that support lightweight interaction and provide DevOps, CI, DI and other features become increasingly prevalent [22].The academic community has conducted in-depth research on tracing, monitoring and fault detecting of applications under the microservice environment [30]. The industry leaders Amazon, Netflix, Facebook, and Uber had executed microservices migration work, and IBM developed an Eclipse IDE plugins for microservice technology to facilitate the development and deployment of microservices, but only for IBM's products. In academia, based on the research of microservice technology architecture, focus on the construction, organization, deployment, quality management, monitoring and error location of microservices [23, 24]. The most related work with ours is [7], the authors construct the monolithic application's class level graph representation based on the source code analysis, and then use semantic, change history and contributor relationship to evaluate the degree of coupling tightness and then extract the candidate microservices through clustering. In [26], the authors used a dataflow-driven method to get the data relationship of the monolithic application, and based that to construct the candidate microservices.

In this paper, we adopts function granularity partition, which compatible with class based partitioning, so it is a more generic solution. From the perspective of program analysis, we combined the static and dynamic analysis result to construct the monolithic application features model; it has a higher accuracy and more extensively applicable scope.

6 CONCLUSION

Monolith architecture application have some bottleneck in terms of scalability, performance, upgrade and maintenance. As the monolith application developed as a whole, so it cannot implements different technology stacks in different modules. In the deployment procedure, there is no way to deploy the monolith application according to the module, which with limited service capability, as it must be deployed as an integral whole. Under monolith architecture condition, we cannot make full use of existing load balancing, different technology stack to improve the development and operational efficiency. After the introduction of the microservices technology architecture, it has attracted wide attention from the academia and industry, which brings an opportunity to solve the above problems of monolith application. In order to prolong the lifecycle of the legacy system, it is need to migrate the traditional monolith application to microservices technology architecture.

In this paper, we presented a program analysis based method to migrate monolith legacy application to microservices architecture. We proposed combining static and dynamic analysis on the application code or binary code to get the characteristics of the legacy monolith application. The function call graph obtained through static analysist of the monolith application always not complete, as the function invoke often defined in the setting file or in the annotation way, and even more the actually actions executed is tightly coupled to user input parameters. We use the dynamic tracing to get the application's features in running, the user input parameters and the functions invoked and the database-accessed information, the frequency of the function invoked etc. al. recorded during the tracing. In this way, we get the application's dynamic features and user input related call relationship. It overcome the inadequacy of static analysis cannot get the application dynamic characteristics. Interaction characteristics in this paper, the application of clustering, extract the behavior characteristics of tightly coupled, through

hierarchical clustering implementation application access features, then based on the segmentation of clustering results to complete the application logic, to generate candidate set of microservices.

In terms of data migration, based on the application of data access and data table structure characteristics of the division of data, by using a single database, through the depots, table and combining with the implementation data columns, rows, divided into division of microservices migration.

In the evaluation section, we compared the differences in performance and scalability between monolith and microservices applications. The effectiveness of this work is verified by experiments and the validity proved by comparing with the existing migration works. The experimental results show that the proposed method can solve the problem of migration of legacy applications efficiently.

ACKNOWLEDGMENTS

This work supported by the National Key Research and Development Plan No.2017YFB1400602 and NSC Discovery Grant No. 61572480. Thanks to the anonymous reviewers advices.

REFERENCES

[1] Microservices. 2018. https://martinfowler.com/articles/microservices.html
[2] A. Balalaie, A. Heydarnoori, P. Jamshidi. 2016. Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. *Advances in Service-Oriented and Cloud Computing ESOCC 2015. Communications in Computer and Information Science*, vol 567.Springer, 201-215.
[3] P. Jamshidi, A. Ahmad, and C. Pahl. 2013. Cloud migration research: A systematic review. *IEEE Transactions on Cloud Computing*, vol. 1, 142-157.
[4] P. Jamshidi, C. Pahl, S. Chinenyeze, and X. Liu. 2015. Cloud Migration Patterns: A Multi-cloud Service Architecture Perspective. *ICSOC 2014, LNCS 8954*, 6-19.
[5] D. Richter, M. Konrad, K. Utecht, and A. Polze. 2017. Highly Available Applications on Unreliable Infrastructure Microservices Architectures in Practice. *International Conference on Software Quality, Reliability and Security (ICSQRS) 2017.* 130-137.
[6] R. C. Martin. 2002. The single responsibility principle. *The Principles, Patterns, and Practices of Agile Software Development*, 149-154.
[7] G. Mazlami, J. Cito, P. Leitner. 2017. Extraction of Microservices from Monolithic Software Architectures. *IEEE 24th International Conference on Web Services (ICWS).* 2017. 524-531.
[8] A. Levcovitz, R. Terra, and M. T. Valente. 2016. Towards a technique for extracting microservices from monolithic enterprise systems. *arXiv preprint arXiv: 1605. 03175* (2016).
[9] S. Newman, 2015. *Bilding Microservices.* O'Reilly Media, Inc.
[10] A. Balalaie, A. Heydarnoori and P. Jamshidi. 2016. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *In IEEE Software*, vol 33, no. 3, 42-52.
[11] A. Balalaie, A. Heydarnoori, and P. Jamshidi. 2015. Migrating to cloud-native architectures using microservices: An experience report. *In European Conference on Service-Oriented and Cloud Computing 2015*, Springer, 201-215.
[12] T. Salah, M. Jamal Zemerly, etc. al. 2016. The Evolution of Distributed Systems Towards Microservices Architecture. *The 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 318-325.
[13] P. Jamshidi, A. Ahmad, C. Pahl. 2013. Cloud Migration Research: A Systematic Review. *IEEE Transactions on Cloud Computing*. Vol 1, No.2, 142-157.
[14] P. Jamshidi, C. Pahl, S. Chinenyeze, X. Liu. 2014. Cloud Migration Patterns: A Multi-cloud Service Architecture Perspective. *The 12th International Conference on Service-Oriented Computing (ICSOC)*, 6-19.
[15] D. Escobar etc.al. 2016. Towards the understanding and evolution of monolithic applications as microservices. *XLII Latin American Computing Conference (CLEI)*, 1-11.
[16] C. Pahl, P. Jamshidi. 2016. Microservices: A Systematic Mapping Study. *Proceedings of the 6th International Conference on Cloud Computing and Services Science*, 137-146.
[17] H. Knoche. 2016. Sustaining Runtime Performance while Incrementally Modernizing Transactional Monolithic Software towards Microservices. *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE)*, 121-124.
[18] C. Fan and S. Ma. 2017. Migrating Monolithic Mobile Application to Microservices Architecture: An Experiment Report. *IEEE International Conference on AI & Mobile Services (AIMS)*, 109-112.
[19] P. Jamshidi, A. Ahmad and C. Pahl. 2013. Cloud Migration Research: A Systematic Review. *IEEE Transaction on Cloud Computing*, Vol. 1, No. 2, 142-157.
[20] P. Jamshidi, C. Pahl, S. Chinenyeze, X. Liu. 2015. Cloud Migration Patterns: A Multi-cloud Service Architectural Perspective. *ICSOC 2014 Workshops,* 6-19.
[21] T. Salah, M. Jamal Zemerly, Chan Yeob Yeun, M. Al-Qutayri and Y. Al-Hammadi. 2016. The Evolution of Distributed Systems Towards Microservices Architecture. *The 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 318-325.
[22] M. Razavian, P. Lago. 2010. A Frame of Reference for SOA Migration. *Towards a Service-Based Internet. ServiceWave 2010*, 150-162.
[23] L. O'Brien, P. Brebner, J. Gray. 2008. Business Transformation to SOA: Aspects of the Migration and Performance and QoS Issues. *Proceedings of the 2nd international workshop on Systems development in SOA environments (SDSOA 2008)*, 35-40.
[24] S. Klock, J. M. E. M. V. D. Werf, J. P. Guelen and S. Jansen. 2017. Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures. *2017 IEEE International Conference on Software Architecture (ICSA)*, 11-20.
[25] S. Hassan, N. Ali and R. Bahsoon. 2017. Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity.*2017 IEEE International Conference on Software Architecture (ICSA)*, 1-10.
[26] R. Chen, S. Li, Z. Li. 2017. From Monolith to Microservices: A Dataflow-Driven Approach, *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 466-475.
[27] S. Li. 2017. Understanding Quality Attributes in Microservice Architecture, *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, 9-10.
[28] N. Alshuqayran, N. Ali and R. Evans. 2018. Towards Micro Service Architecture Recovery: An Empirical Study. *2018 IEEE International Conference on Software Architecture (ICSA)*, 47-56.
[29] X. Zhou, X. Peng, T. Xie, J. Sun, C. Xu, C. Ji, D. Zhou, W. Zhao. 2018. Benchmarking Microservice Systems for Software Engineering Research. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings (ICSE)*, 323-324.
[30] X. Zhou, X. Peng, T. Xie, J. Sun, W. Li, C. Ji, D. Ding. 2018. Delta Debugging Microservice Systems. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 802-807.
[31] M. Gysel, L. Kolbener, W. Giersche, and O. Zimmermann. 2016. Service Cutter: A systematic approach to service decomposition, *In Proc. ESOCC 2016.* Springer, 2016, 185–200.