Implementation of Domain-oriented Microservices Decomposition based on Node-attributed Network

Lingli Cao
Anhui Provincial Key Laboratory of Multimodal Cognitive Computation, Anhui University, China; School of Computer Science and Technology, Anhui University, China; State Key Lab for Novel Software Technology, Nanjing University, China, Email: e19201094@stu.ahu.edu.cn

Cheng Zhang*
Anhui Provincial Key Laboratory of Multimodal Cognitive Computation, Anhui University, China; School of Computer Science and Technology, Anhui University, China; State Key Lab for Novel Software Technology, Nanjing University, China, Email: cheng.zhang@ahu.edu.cn

ABSTRACT

The features of microservices, such as scalability and maintainability, have attracted the industry to migrate monolithic projects to microservices. However, how to decompose microservices during migration is a tricky problem. At present, microservices decomposition mainly relies on architects or domain experts, which is more subjective and time-consuming. Followed by semi-automated or automated microservice decomposition, such methods produce coarse-grained results affected by different system characteristics, which cannot make desired decomposition according to the migration requirements of domains. Therefore, this paper proposes a domain-oriented fine-grained microservices decomposition resolution scheme. It uses dynamic and static analysis to obtain the invocation relationships and invocation times between entity methods and the response time of entity methods to represent three main system characteristics concerned during the migration: function, inter-service communications, and performance. And express the above information of monolith by the node-attributed network. Then it uses the community detection algorithm and the proposed similar hierarchical clustering algorithm to complete objective and effective decomposition. Finally, the rationality and feasibility of the proposed approach are verified using the JPetStore case.

CCS CONCEPTS

• **Software engineering**; • **Software architecture and design**; • **Microservices decomposition**;

KEYWORDS

Microservices Decomposition, Domain, Node-attributed Network, Dynamic analysis, Static analysis

ACM Reference Format:
Lingli Cao and Cheng Zhang. 2022. Implementation of Domain-oriented Microservices Decomposition based on Node-attributed Network. In *2022 11th International Conference on Software and Computer Applications (ICSCA 2022), February 24–26, 2022, Melaka, Malaysia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3524304.3524325

*Corresponding Author

1 INTRODUCTION

In a microservices architecture, each service has a single business capability and can be independently expanded and automatically deployed in its process. Since Fowler and Lewis formally proposed the concept of microservices architecture in 2014 [1], it has attracted the attention of industry and academia. Many companies have successfully applied microservices architecture to practical projects, such as Amazon, eBay, etc. When migrating legacy monolith to microservices, the core of success lies in whether the boundaries and granularity of microservices are properly defined [2]. Inappropriate microservices decomposition can lead to excessive system size and complexity, and huge losses [2]. Currently, the industry still relies on business analysis and Domain-driven Design (DDD) [3] for microservices decomposition, which is subjective and time-consuming [4]. The existing semi-automation and automated microservices decomposition methods ([5], [6], [7], etc.) improve migration efficiency but are limited in use. They can only meet specific system requirements and are only applicable to projects with the same system requirements in the same domain.

In our previous work, we have conducted a systematic literature review (SLR) of the microservices decomposition method between 2014 and 2021. We summarized system characteristics of concern in different domains, mainly including functional requirements, performance, and inter-service communications. To meet the migration requirements of systems in different domains and provide more accurate microservices decomposition, this paper proposes a novel microservices decomposition method based on the node-attributed network from the method granularity. The concept of the node-attributed network is derived from the node-attributed social network which is a data map of a social network in the real world and is generally analyzed using community detection [8]. Community detection is to unsupervisedly mark the closely connected and interrelated network nodes as independent clusters, which are called communities. Traditional community detection methods can be divided into two types: one is clustering based on the structure of the social network (i.e., the topological relationship between nodes); the other is clustering based on the attributes of the social network (i.e., the characteristics of nodes) [8]. However, the results of detecting communities only according to the network structure or attributes cannot fully reflect the real communities' appearance. Therefore, researchers proposed a variety of fusion algorithms, such as [9], [10], to deal with both network structure and attributes when carrying out community detection. We use the dynamic analysis tool kieker [11] and static analysis tool java-callgraph [12] to obtain the invocation relationships and invocation times between entity methods, and the response time (in nanoseconds) of entity methods to represent three main system characteristics concerned during the migration: function, inter-service communications, and performance respectively. And express the above information of monolith by the node-attributed network. Then we use the community detection algorithm and the proposed similar hierarchical clustering algorithm to provide objective and effective microservice candidates recommendations. Finally, the rationality and feasibility of the proposed approach are verified using the JPetStore (https://github.com/mybatis/jpetstore-6) case.

The remainder of this paper is arranged as follows. Section 2 reviews the related work. Section 3 elaborates the proposed microservices decomposition method. Section 4 introduces the evaluation experiments. Section 5 describes the discussion and Section 6 is the conclusion and our future work.

2 RELATED WORK

At present, there are mainly three ways for extracting microservices, which are based on functional requirements, non-functional requirements, or internal characteristics of the monolithic system. Extracting microservices based on functional requirements is the most common way for decomposition. It is mostly done by analyzing graphical documents that express system business, such as use case diagrams, UML class diagrams, data models, business models, etc.. [13]used use case diagrams to decompose legacy systems into functionally independent services. [14] realized the application of microservices architecture by creating a business process model and decomposing it according to the principles of microservices design. Non-functional requirements are the primary migration motivation for monolith migration to microservices. Decision-makers often choose to use microservices architecture when they need to improve non-functional characteristics such as system performance [15], scalability [16], etc. It is a more convenient and efficient method to decompose microservices according to the internal characteristics of the system. By analyzing the dependencies between software system elements (such as static characteristics, runtime dynamic characteristics, etc.) and the communication structure of inter-services, the closely connected system components are identified and mapped to microservice candidates. Based on the system access log, a decomposition framework is proposed, which greatly reduces the subjectivity of the decomposition process [17].

These research efforts are limited to the decomposition of microservices that meets specific requirements. But in a real industry project, the motivation to move to a microservices architecture is often multifaceted. Different application domains, development requirements are not the same. That is, these methods cannot be used as generalized approaches. Therefore, the proposed method combining the characteristics of function, performance, and inter-service communications, transforms the problem of microservices decomposition into the community detection of the network. Thus, it simplifies the difficulty of microservices identification while realizing domain-oriented microservices decomposition.

3 METHODOLOGY

The proposed microservices decomposition process based on the node-attributed network is shown in Figure 1. To obtain comprehensive and detailed monolith information, we take dynamic analysis as the main, static analysis as the auxiliary way to analyze the monolithic source code for collecting the entity methods behavior characteristics of the monolith. The above data are used to construct the node-attributed network corresponding to the monolith, and then the topological structure and attribute characteristics are fused for converting into the form applicable to the classical community detection algorithm [18], to obtain the domain-oriented preliminary microservice candidates. Finally, referring to the high cohesion and low coupling of the microservices design principle [19] and hierarchical clustering algorithm [20], we propose a similar hierarchical clustering algorithm to optimize the preliminary microservice candidates and obtain the final microservices decomposition result.

3.1 Collect The Monolith Information on Method Granularity

This section describes how to use dynamic and static analysis to obtain behavioral characteristics of the monolithic entity methods, including invocation relationships and times between methods, and response time of methods (see Figure 2). Information of empirical case JPetStore is used to demonstrate data format changes during the proposed method execution.

3.1.1 Dynamic Analysis. This paper strictly follows the kieker user guide to configure the monolith for monitoring the monolith in operation. If the monolith does not have automated test components, we also need to design additional test cases that cover as fully as possible the system business to help generate monitoring logs of system behavior characteristics at the method granularity. As shown in Figure 2a, each row of the monitoring log consists of 11 elements, which are the operation type of record, sequence ID, monitoring probe type, the class and method called in the current test case (CM), session ID, trace ID, the start time (Tin) of the calling method, and the end time (Tout), hostname, method execution order index (Eoi), method execution stack size (Ess). And then, CM, trace ID, Tin, Tout, Eoi, and Ess are used to generate a message trace which describes the dynamic behavior of the system in an orderly message sequence [11]. After deleting contents unrelated to system business, we recorded the invocation times ***n*** of invocation relationship ***r*** in the message trace, and the average response time ***t1*** and ***t2*** of entity methods ***m1*** and ***m2*** in ***r*** respectively, thus forming the final dynamic behavior characteristic data of entity methods, where each row consists of ***m1***, ***t1***, ***m2***, ***t2***, ***n***.

3.1.2 Static Analysis. This paper strictly follows the java-callgraph user guide to analyze the monolith. As shown in Figure 2b, the analysis text consists of class and method granularity monolith invocation information respectively. The proposed method only uses method-granularity invocation relationships, that is, rows beginning with M, which consists of the entity method invocation

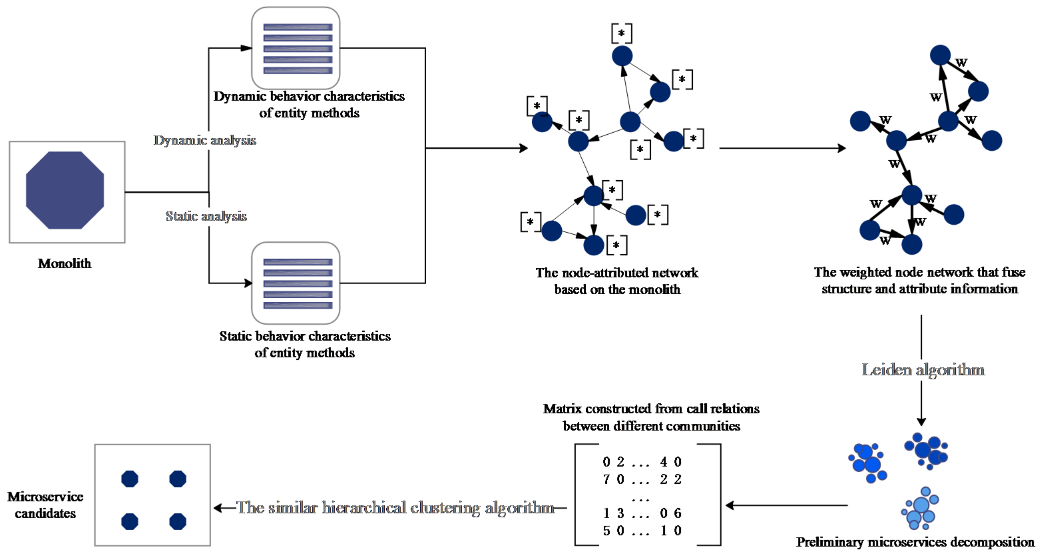
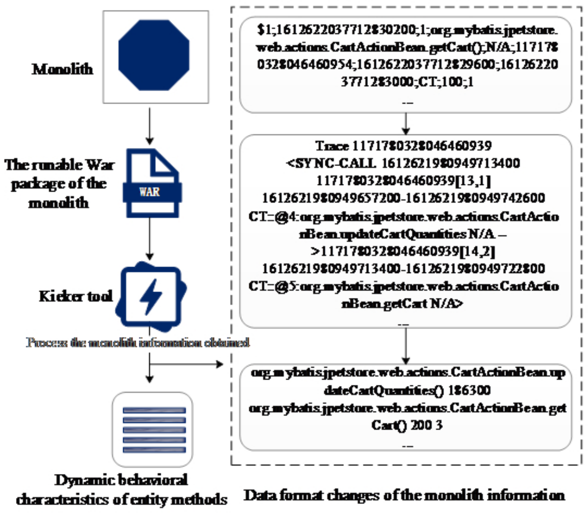

Dynamic behavior characteristics of entity methods
Dynamic analysis
Static analysis
Monolith
Static behavior characteristics of entity methods
The node-attributed network based on the monolith
The weighted node network that fuse structure and attribute information
Leiden algorithm
Matrix constructed from call relations between different communities
0 2 ... 4 0
7 0 ... 2 2
...
1 3 ... 0 6
5 0 ... 1 0
Preliminary microservices decomposition
The similar hierarchical clustering algorithm
Microservice candidates

Figure 1: The overall process of executing the proposed microservices decomposition method.

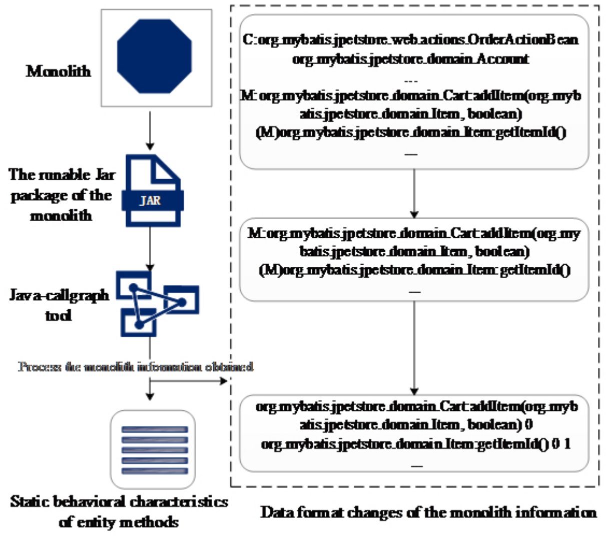

Monolith
The runable War package of the monolith
WAR
Kicker tool
Process the monolith information obtained
Dynamic behavioral characteristics of entity methods
Data format changes of the monolith information

```
$1;1612622037712830200;1;org.mybatis.jpetstore.
web.actions.CartActionBean.getCart();N/A;117178
0328046460954;1612622037712829600;16126220
37712883000;CT;100;1
...

Trace 1171780328046460939
<SYNC-CALL 1612621980949713400
1171780328046460939[13,1]
1612621980949657200-1612621980949742600
CT::@4:org.mybatis.jpetstore.web.actions.CartActio
nBean.updateCartQuantities N/A –
>1171780328046460939[14,2]
1612621980949713400-1612621980949722800
CT::@5:org.mybatis.jpetstore.web.actions.CartActio
nBean.getCart N/A>
...

org.mybatis.jpetstore.web.actions.CartActionBean.up
dateCartQuantities() 186300
org.mybatis.jpetstore.web.actions.CartActionBean.get
Cart() 200 3
...
```

(a) Dynamic analysis


Monolith
The runable Jar package of the monolith
JAR
Java-callgraph tool
Process the monolith information obtained
Static behavioral characteristics of entity methods
Data format changes of the monolith information

```
C:org.mybatis.jpetstore.web.actions.OrderActionBean
org.mybatis.jpetstore.domain.Account
...
M:org.mybatis.jpetstore.domain.Cart:addItem(org.myb
atis.jpetstore.domain.Item, boolean)
(M)org.mybatis.jpetstore.domain.Item:getItemId()
...

M:org.mybatis.jpetstore.domain.Cart:addItem(org.my
batis.jpetstore.domain.Item, boolean)
(M)org.mybatis.jpetstore.domain.Item:getItemId()
...

org.mybatis.jpetstore.domain.Cart:addItem(org.myb
atis.jpetstore.domain.Item, boolean) 0
org.mybatis.jpetstore.domain.Item:getItemId() 0 1
...
```

(b) Static analysis

Figure 2: The process of analyzing the monolith.

relationship and invocation type. Referring to the treatment of dynamic behavior characteristics of entity methods, we added entity methods response time (value set to 0) and invocation times (value set to 1) for each row and delete unrelated information to keep the two data formats consistent. Each row of the final static behavior characteristic data of entity methods is composed of $m1$, 0, $m2$, 0, and 1.

3.2 Construct The Node-attributed Network of The Monolith

Combining the dynamic and static behavior characteristic data of the monolith entity methods, we construct the corresponding node-attributed network G. Each entity method m is mapped to the node v in G, the response time t of the method is mapped to the attribute Va of the nodes, representing the performance characteristics of the monolith; the invocation relationships R between the methods are mapped to the directed edge E in G, representing the business of the monolith, and the invocation times between entity methods are mapped to the attribute Ea of E, representing the communication overhead of the monolith. Before detecting community, we fuse the structure and attributes of the node-attributed network to construct a weighted network Gw that can replace G and make it suitable for the classical community detection algorithm [8]. Referring to [10], the weight of the edge in Gw is the product of the completion time of R and Ea, where the completion time of each invocation relationship is the attribute value of the called method in Va. And then we use the Leiden algorithm [18] to conduct community detection on Gw and generate preliminary microservice decomposition results.

3.3 Identify Microservice Candidates by The Proposed Similar Hierarchical Clustering Algorithm

The microservices design principles specifically emphasize that microservices have high cohesion and loose coupling [19]. After the preliminary microservice decomposition results are obtained in Section 3.2, this proposed method yields a microservice decomposition scheme. But this solution is not necessarily the best, and in actual industrial practice, there are often insufficient human resources to form microservices development teams. Therefore, we use the idea of a hierarchical clustering algorithm [20] to propose a similar hierarchical clustering algorithm (see Algorithm 1) to optimize it, and flexibly control the number of microservice candidates, to produce a more reasonable and feasible microservice decomposition scheme.

We first used the preliminary microservices decomposition results to create a matrix representing the number of invocation relationships between different communities. It is a symmetric matrix, so we'll only deal with the upper right triangle. We walk through the matrix row by row, get the column index with the maximum value per row, and merge the row and column as a cluster. In this process, if the right and under element values of the current row is 0, then the community represented by the row is an isolated community, which is merged as a cluster to save development resources. For clustering results generated after traversal, if there are subsets between different clusters, the subsets will be deleted; if there is overlap between different clusters, the optimal cluster will be selected as described in Algorithm 1. Repeat those steps until the cluster number matches the number of microservice candidates.

Algorithm 1 The similar hierarchical clustering algorithm

```
Input: Number of microservice candidates: n; Preliminary microservice decomposition results: p_partition
Output: Optimized microservice candidate results
Create a community association matrix: Matrix
row=0                // row represents the row number of the Matrix
total_row            // total_row represents the total row number of the Matrix
while row < total_row, do
    col=row+1        // col represents the col number of the Matrix
    Merge isolated communities as a cluster
    Returns the col with the maximum value in the current row
    Add row and col as a new community to the array p
// p was used to save the clustering results
end
if there have subsets between the elements in p, do
    delete the subsets
end
if there is a partial duplication between the elements in p, do
    remove the element that contain fewer invocation relationships
    if the number of invocation relationships is the same, do
        remove the element that contain a greater number of the original communities
        if the number of original communities is equal, do
            remove the element positioned backward in p
        end
    end
end
repeat the above steps until the number of elements in p is n
return p
```

4 EVALUATION

This paper uses a classic open-source project JPetStore (https://github.com/mybatis/jpetstore-6) to validate the proposed method, which is a web application based on Mybatis3, Spring 3, and Stripes. It consists of 2059 lines of code and 24 classes. The business layer of this project is not clear. In case of a sudden increase in business, its scalability and maintainability are low and it is suitable for migrating to microservices. Therefore, it is often used as a verification project of microservices decomposition methods [6, 7]. Selecting it as experimental data and comparing it with existing microservice decomposition methods can verify the rationality and effectiveness of the proposed method.

4.1 Experiments Design

Experiment I - The proposed method to extract microservices: Using this method, we obtain domain-oriented microservice candidates.

Experiment II - Function-oriented microservice candidates identification (FoSCI) [6]: This method considers the decomposition of microservices at the class level. It focuses on analyzing a certain class of operation or system business. In contrast, our proposed approach is based on the method level. Therefore, our method focuses on analyzing specific operations or system businesses for obtaining detailed information about the system to achieve more accurate decomposition. That is, the two approaches have different views on service granularity. Therefore, we need to refine the decomposition results to the method level for comparative evaluation after applying the FOSCI method to get microservice candidates.

Experiment III - Data flow-driven method (DFD) to extract microservices [21]: Similar to the method in Experiment II, the DFD method also considers the service granularity from the class level. Therefore, we perform the same operation as in Experiment II. First, we expand the decomposition results, list the method entities contained in each class for obtaining more fine-grained microservices, then perform the corresponding quality measures.

Experiment IV - Using the distributed representation of source code to extract microservices [7]: This method uses a neural network model (code2vec) to create code embeddings from the source code of the monolith, and then uses a hierarchical clustering algorithm to perform clustering on semantically similar code embeddings to generate microservice candidates.

Experiment V - Manual decomposition to extract microservices: This method is used as the baseline. The operator has more than 3 years of software system development experience. First, we analyze and summarize the business functions of the monolith. Then we use the breakpoint method to obtain the entity methods information that realizes the business function. Finally, the obtained entity methods sets are checked and collated to obtain the determined microservice candidates.

4.2 Evaluation Metrics

In the area of microservices architecture, a unified microservice quality evaluation standard has not yet been produced. By studying the existing microservice quality assessment work and referring to the classical measurement methods of internal and external connectivity of modular components of the software system [22], we proposed microservices cohesion and coupling measurement metrics M_Ch (cohesion in method-level) and M_Cp (coupling in method-level) based on method granularity (see Equations 1 and 2). In order to evaluate the quality of microservice candidates as a whole, the Density evaluation metric [23] is also included. Let us briefly explain some of the parameters that exist in metrics, and then introduce each of them.

M_M_i is the number of all methods contained in $Service_i$, $M_E_i$is the number of all edges contained in $Service_i$, $Services = \bigcup_{i=1,2,3,\ldots,k} Service_i$, $Service_i = (M\_M_i, M\_E_i)$, E is the number of all edges in the Services.

$$M\_Ch = \frac{1}{k}\sum_{i=1}^{k}\frac{M\_E_i}{M\_M_i^2} \tag{1}$$

$$M\_Cp = \frac{1}{k(k-1)/2}\sum_{i\neq j}^{k}\frac{M\_E_{i,j}}{2 * M\_M_i * M\_M_j} \tag{2}$$

$$Density = \frac{1}{|E|}\sum_{i=1}^{k}|M\_E_i| \tag{3}$$

M_Ch is a variant of connectivity within system components. The value of M_Ch is between 0 and 1. The higher the value of M_Ch, the higher the quality of microservice candidates. M_Cp is a variant of inter-connectivity between system components. The value of M_Cp is between 0 and 1. The smaller the M_Cp value, the higher the quality of the microservice candidates. Density is used to indicate the internal correlation degree of each microservice candidate. The higher the value of the density represents the better the quality of the microservice candidates.

4.3 Results

Compared with the manual decomposition method (Experiment V) and three mainstream representative microservices decomposition methods acknowledged by researchers (Experiments II-IV), we verify that our proposed method (Experiment I) can produce high-quality microservice candidates.

As shown in Table 1 and Figure 3, the proposed method has achieved a great improvement over the manual decomposition method (Experiment V) in all metrics. Although they can all be domain-oriented decomposition, the manual method is very subjective, and the proposed method can make an objective and efficient microservices decomposition. For example, when a monolithic system in the information service domain migrates to microservices architecture, it is necessary to ensure that the communication structure of the system is maintained well. If the manual decomposition method is adopted, it is impossible to give a precise definition to the communication of the system. Using the proposed method, we can understand the communication trace of the system in detail, to carry out accurate microservices decomposition.

It can be seen from Table 1 that Experiments I, II, and III show the same superiority on the M_Cp metric, indicating that these three methods can effectively achieve decoupling between microservices. Because they are mainly based on the system business, they can realize the independence of service responsibilities. But in terms of the remaining metrics, the proposed method is significantly better than the other two methods. It decomposes microservices from method granularity, so it can obtain comprehensive and detailed system information, thereby achieving accurate decomposition. Experiment IV realizes the decomposition from the perspective of system internal characteristics, but cannot obtain the fine-grained system business. Therefore, the generated microservice candidates have a certain degree of coupling while achieving high cohesion.

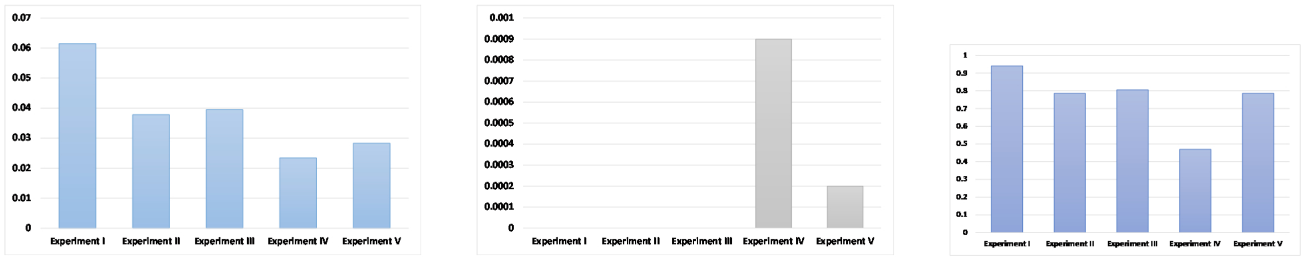
Due to space limitations, the experiment data and the decomposition results have been packaged and placed on the GitHub code hosting platform (https://github.com/microservices2020/MS-decomposition) for follow-up work.

5 DISCUSSION

By performing the contrast experiments, it can be confirmed that the proposed method can provide effective microservice candidates. This is mainly because our method analyzes the monolith information at the method entity level, so a more detailed business description can be obtained, and the decomposition of microservices can be discussed more comprehensively. And in actual industrial practice, there are often insufficient human resources to form microservices development teams. Thus, we propose a similar hierarchical clustering algorithm to optimize and flexibly control microservices decomposition so that promote its application of the proposed method in industrial projects.

Table 1: Quality assessment results

Experiments\Metrics	M_Ch	M_Cp	Density
Experiment I	0.0615	0.0000	0.9408
Experiment II	0.0378	0.0000	0.7855
Experiment III	0.0395	0.0000	0.8058
Experiment IV	0.0234	0.0009	0.4696
Experiment V	0.0283	0.0002	0.7855



(a) The results on M_Ch **(b) The results on M_Cp** **(c) The results on Density**

Figure 3: The results of evaluation

The proposed method can better meet the migration requirements of different domains. Although other methods can also produce good results compared with manual decomposition, they can only meet specific requirements. The FoSCI method mainly divides the system from the perspective of business functions, which can quickly realize the reconstruction from monolith to microservices. But it ignores that the longest phase in the life cycle of a software system is maintenance and expansion. Besides, systems in different domains have different characteristics during operation and maintenance. So, the system characteristics should be considered when decomposing for efficiently applying microservices architecture. When using the DFD method, it is first necessary to manually draw the data flow diagram of the legacy monolith. This subjective process determines the result of microservices decomposition. In addition, this method needs to be combined with the documents, such as design instructions of the monolith, otherwise, it will not be able to generate a definite system data flow diagram. However, our method is based on the analysis of the monolith source code, even if the documentation is missing, it can still accurately reflect the system information without affecting the decomposition of microservices. As for the code2vec method, it focuses on analyzing semantic similarity, and then divides the microservices using a hierarchical clustering algorithm. This method is highly dependent on the monolith developed under standard code specifications that would otherwise produce poor microservice candidates. In contrast, our method only needs the monolith to be in a normal operation state, and can quickly provide an effective decomposition solution for refactoring the monolith into microservices.

When adjusting microservice candidates, we ensure high cohesion within microservices while reducing the coupling and interaction between services as much as possible. If the architect is satisfied with the decomposition result produced by this method, development work can be executed quickly. If the architect is partially satisfied with the decomposition result, the architect can make corrections. Therefore, our method can help architects improve the quality and efficiency of their work and avoid unreasonable decomposition.

In the initial stage of method execution, we need to design test cases that cover the legacy system business functions as comprehensively as possible to obtain accurate system dynamic behavior characteristics. There will be a certain degree of subjectivity in this process, but in the actual project, the person in charge has enough knowledge of the target system and can objectively design test cases. This relatively reduces the impact of this threat. Second, since this paper discusses dynamic and static data characteristics separately when constructing the node-attributed network, there will be some overlapping of method entities in the microservice candidates. For achieving non-overlapping decomposition, i.e., independent microservices, we choose to use dynamic data as a benchmark and eliminate duplication in static data. That is because dynamic data has more detailed method entity characteristics that can better reflect system behavior.

6 CONCLUSION AND FUTURE WORK

In our previous work, we conducted an empirical study on microservices decomposition and confirmed that the monoliths in different domains have different system requirements when migrating. The existing microservices decomposition methods do not consider this, and can only handle some decomposition tasks for specific requirements. Therefore, it is likely that suitable service decomposition methods will not be found for different domains. On this basis, this paper proposes a novel domain-oriented microservices decomposition method based on the node-attributed network. Experimental results show that the proposed method is effective.

In future work, we will no longer be limited to method-level granularity, and build a node-attributed network that includes packages, classes, and methods from the perspective of multi-level abstraction. The primary business is mapped to fine-grained (methods) network nodes, and coarse-grained (classes, packages) network nodes that are created for non-critical business to better match the legacy monolith information.

ACKNOWLEDGMENTS

This work is supported by the Anhui Provincial Natural Science Foundation (2008085MF189), the Anhui Provincial Key Research and Development Program (202104d07020008).

REFERENCES

[1] J. Lewis, and M. Fowler, "Microservices: A definition of this new architectural term," 2014; https://martinfowler.com/articles/microservices.html.
[2] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," Journal of Systems and Software, vol. 146, pp. 215-232, 2018.
[3] E. Evans, Domain-driven design: Tackling complexity in the heart of software: Addison-Wesley Professional, 2004.
[4] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," Journal of Systems and Software, vol. 182, pp. 111061, 2021.
[5] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kröger, "Microservice decomposition via static and dynamic analysis of the monolith," in 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), 2020, pp. 9-16.
[6] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identi-fication from monolithic systems based on execution traces," IEEE Transactions on Software Engineering, vol. 47, no. 5, pp. 987-1007, 2021.
[7] O. Al-Debagy, and P. Martinek, "A microservice decomposition method through using distributed representation of source code," Scalable Computing: Practice and Experience, vol. 22, no. 1, pp. 39-52, 2021.
[8] P. Chunaev, "Community detection in node-attributed social networks: A survey," Computer Science Review, vol. 37, pp. 100286, 2020.
[9] J. D. Cruz, and C. Bothorel, "Information integration for detecting communities in attributed graphs," in 2013 Fifth International Conference on Computational Aspects of Social Networks, 2013, pp. 62-67.
[10] Y. Ruan, D. Fuhry, and S. Parthasarathy, "Efficient community detection in large networks using content and links," in Proceedings of the 22nd international conference on World Wide Web, Rio de Janeiro, Brazil, 2013, pp. 1089–1098.
[11] A. v. Hoorn, J. Waller, and W. Hasselbring, "Kieker: A framework for application performance monitoring and dynamic software analysis," in Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, Boston, Massachusetts, USA, 2012, pp. 247–248.
[12] X. Zhang, Y. Zhou, and C. Zhu, "An empirical study of the impact of bad designs on defect proneness," in 2017 International Conference on Software Analysis, Testing and Evolution (SATE), 2017, pp. 1-9.
[13] C. Richardson, "Microservices: Decomposing applications for deployability and scalability," InfoQ, vol. 25, pp. 15-16, 2014.
[14] P. Yugopuspito, F. Panduwinata, and S. Sutrisno, "Microservices architecture: Case on the migration of reservation-based parking system," in 2017 IEEE 17th International Conference on Communication Technology (ICCT), 2017, pp. 1827-1831.
[15] S. Eski, and F. Buzluca, "An automatic extraction approach: Transition to microservices architecture from monolithic application," in Proceedings of the 19th International Conference on Agile Software Development: Companion, Porto, Portugal, 2018, pp. 1-6.
[16] P. Martinek, and O. Al-Debagy, "A new decomposition method for designing microservices," Periodica Polytechnica Electrical Engineering and Computer Science, vol. 63, no. 4, pp. 274-281, 2019.
[17] D. Taibi, and K. Systä, "From monolithic systems to microservices: A decomposition framework based on process mining," in Proceedings of the 9th International Conference on Cloud Computing and Services Science - CLOSER, 2019, pp. 153-164.
[18] V. A. Traag, L. Waltman, and N. J. van Eck, "From louvain to leiden: Guaranteeing well-connected communities," Scientific Reports, vol. 9, no. 1, pp. 1-12, 2019.
[19] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in 2017 IEEE International Conference on Software Architecture (ICSA), 2017, pp. 21-30.
[20] F. Murtagh, and P. Contreras, "Algorithms for hierarchical clustering: An overview," WIREs Data Mining and Knowledge Discovery, vol. 2, no. 1, pp. 86-97, 2012.
[21] S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, J. Ge, and Z. Shan, "A dataflow-driven approach to identifying microservices from monolithic applications," Journal Of Systems And Software, vol. 157, pp. 110380, 2019.
[22] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner, "Using automatic clustering to produce high-level system," in Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242), 1998, pp. 45-52.
[23] C. Bothorel, J. D. Cruz, M. Magnani, and B. Micenkova, "Clustering attributed graphs: Models, measures and methods," Network Science, vol. 3, no. 3, pp. 408-444, 2015.