
Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2021 SCPE. Volume 22, Issues 1, pp. 39–52, DOI 10.12694:/scpe.v22i1.1836

A MICROSERVICE DECOMPOSITION METHOD THROUGH USING DISTRIBUTED REPRESENTATION OF SOURCE CODE

OMAR AL-DEBAGY* AND PÉTER MARTINEK

Abstract. This research proposed a novel decomposition method for refactoring monolithic applications into microservices applications using a neural network model (code2vec) for creating code embeddings from the monolithic application source code. As a Result, semantically similar code embeddings are clustered through a hierarchical clustering algorithm to produce microservices candidates to resemble the domain model more efficiently. The quality characteristics of the results were measured using two metrics for measuring cohesion. These metrics were Cohesion at Message Level (CHM) and Cohesion at Domain Level (CHD). Also, four applications were used as test cases with different sizes ranging from small to big applications. The proposed method showed promising results in terms of cohesion when compared to other decomposition methods. The proposed method scored better scores in 5 out of 8 tests compared to other methods. Also, averaged CHD and CHM results were 0.52 and 0.76, respectively, for the proposed method, better results when compared to the other methods.

Key words: microservices decomposition, microservices, refactoring

AMS subject classifications. 68M14

1. Introduction. Nowadays, the internet requires a more flexible, scalable, and understandable software architecture. Therefore, many companies and organizations started the process of migration from the monolithic architecture toward a more suitable architecture that meets the demands of the current market [1]. The current market requires an architecture flexible enough to face the frequent changes in user demands, and easily scalable architecture to face the massive number of users [2]. These reasons led many companies and organizations to adopt the microservices architecture. They chose microservices architecture to have a more scalable, easier to maintain, and easier to manage applications [3]. Microservices is an architecture of fine-grained services that are working independently from each other and communicating with each other through lightweight mechanisms to do the tasks of an application suite. Although microservices provide different advantages to the organization, but the process of migration introduced multiple issues. One of these issues is how to decompose or refactor an existing monolithic application into microservices, which are loosely coupled and highly cohesive at the same time, according to Newman [4]. Multiple researchers have introduced several methods to decompose the monolithic application [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. These methods include many different approaches such as static analysis of the application's code [8, 9, 10, 14], dynamic analysis of the performance of the applications [11, 12, 15, 16], and analysis of the applications interfaces [7, 17, 13]. These methods provide assistance for developers to help them in the process of migrating from monolithic applications to the microservices application. Still, the result of these methods cannot be considered as absolute results. This research aims to tackle the issue of microservices identification using vector representation of software's source code. Hence, this paper proposed a novel approach to decompose monolith application into a microservices application by using a neural model [22] to represent snippets of code as continuous distributed vectors. The code of the monolithic application would be converted into vector representation using the provided model, and then certain classes would be grouped to provide microservices candidates.

Four monolithic applications with different sizes were decomposed using the provided methodology in order to verify the effectiveness of this approach. These applications were tested in other research papers before, so they are considered benchmark applications for monolithic applications' decomposition process. We used two different metrics to compare the proposed method's performance with other methods from the literature. Also,

*Department of Electronics Technology, Budapest University of Technology and Economics, Budapest, Hungary (omeraldebagy@gmail.com)

we utilized other metrics to compare the sizes of the mentioned applications in the test, such as the number of classes, number of methods, lines of code (LoC), and the number of microservices. The proposed method is a useful aiding tool for developers in the process of migration from monolithic to a microservices architecture, which suggests a specific direction for the decomposition process.

The goal of this paper is to investigate the effectiveness of using code embeddings in the process of decomposing monolithic applications into microservices ones. Also, provide a new technique for extracting microservices from monolithic applications.

The rest of this research is organized as follows: the literature review provides different approaches for handling microservices decomposition. Then, the methodology section presents the details of the proposed methodology and how the decomposition method was constructed. After the methodology, there is a section with the results and the discussion. Finally, a conclusion section highlights future potentials for the method and other possible implementations.

2. Literature Review. There are several research papers related to microservices identification or decomposition. These researches provide different types of methods that can be divided into three groups. The first group is based on the static analysis of the source code of the monolithic application. The second group is based on the dynamic analysis of the monolithic application. Finally, the third group is using the analysis of the application program interfaces or application interfaces.

Abdullah et al. [15] created a decomposition method that considers the scalability and performance of the application and improve its performance after decomposition. Their method used an unsupervised machine learning approach analysing access logs of monolithic applications. Then, they proposed a method to automatically assign the type of virtual machines and their resources to the microservices instances on a cloud architecture. Their method of decomposing a monolithic application based on the application's performance can be misleading because it depends on how the application can be used or how the users are using it. Therefore, the methods based on performance analysis need to have very detailed testing scenarios to work efficiently.

Mazlami et al. [9] proposed a decomposition method for monolithic applications by analysing the version control repository of the application and converting it into graphs for detecting microservices candidates using a graph clustering algorithm. Their method consisted of three different extraction strategies, which are Logical Coupling Strategy, Semantic Coupling Strategy, and Contributor Coupling Strategy. One limitation of the method is the use of classes without considering methods and their input and output parameters.

Kamimura et al. [8] created a method for extracting microservices candidates from source code using a clustering algorithm. They tested their method on two different applications, and two developers reviewed their results. Also, they visualized the provided microservices for the ease of understanding with the Software Architecture Finder (SArF) map for visualization.

Li et al. [6] proposed a data-flow driven approach for decomposing monolith applications into microservices candidates. They highlighted how the decomposition process is different between service-oriented architecture (SOA) and Microservices. First, services in SOA are coarse-grained while in microservices are fine-grained. Second, the process is bottom-up in SOA and top-down first then bottom-up in microservices. Their method consists of 4 steps, first analysing requirements of the monolithic application. Second, constructing data flow diagrams (DFD). Third, compress DFDs into decomposable DFDs. Fourth, propose microservice candidates through decomposable DFDs.

Furthermore, they used cohesion and coupling metrics to evaluate their results compared to Service Cutter and API analysis. The issue with this approach is the need for attention to details in order to create a detailed DFD to make the process of identifying appropriate microservices. Furthermore, they used a relatively small application for the evaluation.

Taibi and Systa [11] proposed a decomposition method using a data-driven approach based on process mining by utilizing log files as a data source. Their decomposition method consisted of 6 steps. The first step is the execution analysis path; the second step is the frequency analysis of the execution path. Removing circular dependencies is the third step. The fourth step is identifying decomposition options. The fifth step is ranking the decomposition options based on metrics. Finally, the sixth step is selecting the decomposition option. They used coupling and the number of classes as metrics for step five. Their evaluation method of depending on the coupling metric is lacking because their method needs other metrics such as cohesion.

Table 2.1: Literature Review Summary

Research	Input	Decomposition Method	Year	Tested Application
Abdullah et al. [15]	Log Files	Performance Based	2019	ACME Air
Mazlami et al. [9]	Commits	Version Control Analysis with Graphs	2017	Multiple
Kamimura et al. [8]	Source Code	SArF software clustering algorithm	2018	PetClinic
Li et al. [6]	Data Flow Diagrams	Analyzing DFD	2019	Cargo App
Taibi and Systa [11]	Log Files	Analysis of the execution paths	2019	Industrial App
Saidani et al. [14]	Source Code	nondominated sorting genetic algorithm	2019	JPetstore, Spring Blog
Jin et al. [12]	Log Files		2018	JPetstore, SpringBlog, JForum, Roller
Nunes et al. [10]	Source Code	Clustering call graphs	2019	LdoD, Blended Workflow
Service Cutter [17]	System Specification Artifacts	Clustering of graphs	2016	Cargo App
Al-Debagy and Martinek [7]	API	Clustering of Operations	2019	Cargo App, Kanban Board, Money App
Santos and Silva [5]	Method Calls	Clustering call graphs	2020	LdoD, Blended Workflow, FenixEdu

Saidani et al. [14] introduced a new decomposition method called MSExtractor. They used the source code of monolithic applications to extract classes and group classes to create microservices candidates. For the evaluation of their method, they used cohesion and coupling metrics. Furthermore, they are utilized a non-dominated sorting genetic algorithm to identify microservices from the source code. This research is compared to the proposed method of this research paper.

Jin et al. [12] proposed a functional oriented decomposition method for microservices applications that monitor the application dynamic behaviour and clusters execution logs or traces. They proposed some evaluation metrics for cohesion and coupling. The logs are generated using specific test cases, but sometimes these test cases cannot cover all the business functionalities, which may lead to ignoring some essential classes and functionalities. The metrics proposed in Jin et al. are used in this paper in order to compare the performance of the proposed method to other decomposition methods.

Nunes et al. [10] developed a method that converts the source code of a monolithic application into call graphs. After that, domain entities are identified, and a clustering algorithm will group these entities. This work has several limitations such as, it is developed for a specific web application framework, and the tool that creates call graphs does not work correctly with all Java versions.

Service Cutter [17] is a decomposition tool that uses domain models and use cases to extract coupling information. This coupling information was defined by the authors and was represented as a weighted graph using Epidemic Label Propagation clustering algorithms.

Al-Debagy and Martinek [7] introduced a decomposition method that relies on the monolithic application's API. They used API operation names to identify the microservices through grouping semantically similar operation names using a hierarchical clustering algorithm.

Santos and Silva [5] proposed a decomposition method that collects graph calls of the monolithic application and converts them into domain entities. Then a similarity function measures the similarity between two entities, and a clustering algorithm groups similar entities together to create microservices candidates. Also, they proposed a complexity metric to verify the validity of the suggested microservices candidates.

Table 2.1 summarizes the methods mentioned in the literature review section 2 and included the applied types of inputs and the type of decomposition they used.

3. Methodology. Machine learning for code refactoring was used on several other software architectures before [23, 24, 25]. However, it can be applied in a microservices' environment as well. This research proposes a new decomposition method for decomposing monolithic applications into microservices applications as follows. The approach uses a novel approach for microservice decomposition by using code representation to understand the similarity within the application classes and cluster semantically similar classes together to create microservices candidates. Clustering semantically similar classes together in order to resemble the domain model more efficiently.

The proposed machine learning-based method consists of these steps:

1. extracting the methods and its code from the monolithic application,
2. converting the code to code embeddings or vector representations,
3. aggregating the code embeddings of one class,
4. group together semantically similar classes to obtain microservices candidates.

3.1. Extracting Code Embeddings. Methods are extracted from classes and converted into code embeddings using the code2vec [22] model. Code embeddings are snippets of codes characterized as a vector-based representation for a machine-learning algorithm to understand these snippets of codes.

Embeddings are a mapping of an object represented as vectors. For example, word embeddings are representations of a word (or sequence of words) as vectors of real numbers [26]. Word embeddings make it possible for textual data to work with a mathematical model. Code embeddings have a similar benefit to word embeddings; these embeddings can capture the semantics of the source code. These code embeddings can be used for several tasks such as malware detection, author identification, and refactoring.

3.2. Code Embeddings Model. The proposed method uses the code2vec model created by Alon et al. [22] to obtain code embeddings or continuous distributed vectors of the extracted methods. Code embeddings give us the ability to find a similarity between the extracted classes.

Code2vec is a deep representation learning method, which was used for predicting method names. However, code2vec code embeddings can be used in other tasks as well. Code2vec converts the source code into a set of Abstract Syntax Tree (AST) paths and sums these paths using an attention mechanism. The attention technique works by giving more weight for the important AST paths that represent the source code. So, the vector representation of a function is an aggregation of weighted AST paths. The attention mechanism shows the important AST paths that need more focus than the other available paths.

AST is represented with branches and leaves similar to a tree. The functional structure of source code is represented by AST instead of a detailed description of source code. For example, Fig. 3.1 shows an AST representation of a factorial function. The utilization of AST improve the accuracy and training of a machine learning model [26].

The goal of code2vec is to generate code embeddings that keep the semantics of the source code. Code2vec represent the source code as a bag of AST paths; these paths are generated between the leaves of the AST tree. AST path is a path between two leaves in an AST tree. For example, the coloured paths in Fig. 3.1 are AST paths. Path-context is a set of three tokens, consisting of two tokens represent the two AST leaves and another token represent the path between these two leaves. For example, the red path in Fig. 3.1 can be represented as follows:

$$\{n, Times \downarrow MethodCall \downarrow Minus \downarrow, n\}$$

The sign $\downarrow$ represent the path goring toward the leaves while $\uparrow$ represent going toward the root of the AST tree. For more details and information check the original paper [22]. Fig. 3.2 shows the architecture of code2vec model with all the processes described earlier.

3.3. Aggregation Method. This step combines the code embeddings of the methods in order to reflect the representation of the class. Multiple aggregation functions were used, such as mean, sum, maximum, minimum, standard deviation, and variance. The mean function gave the best results regarding the accuracy of the clustering function in the next step. Fig. 3.3 shows the process of aggregating multiple code embeddings into one vector representation using the mean function. Table 3.1 lists all the aggregation methods that we tested to find the most applicable aggregation method for the proposed decomposition algorithm.

After this step, the aggregated code embeddings are sent to the next step, which is the clustering method, where it will generate the microservices candidates.

3.4. Clustering Method. Following the conversion of the source code into code embeddings based on code2vec model and aggregating code embeddings, a clustering method was applied. Related classes are clustered together using the clustering method in order to generate a suitable microservice candidate. The Affinity Propagation [19] algorithm was chosen for this process because it identifies the sum of clusters minus the necessity to indicate it in advance. Microservices candidates are identified using the previously mentioned methods combined with the clustering algorithm.

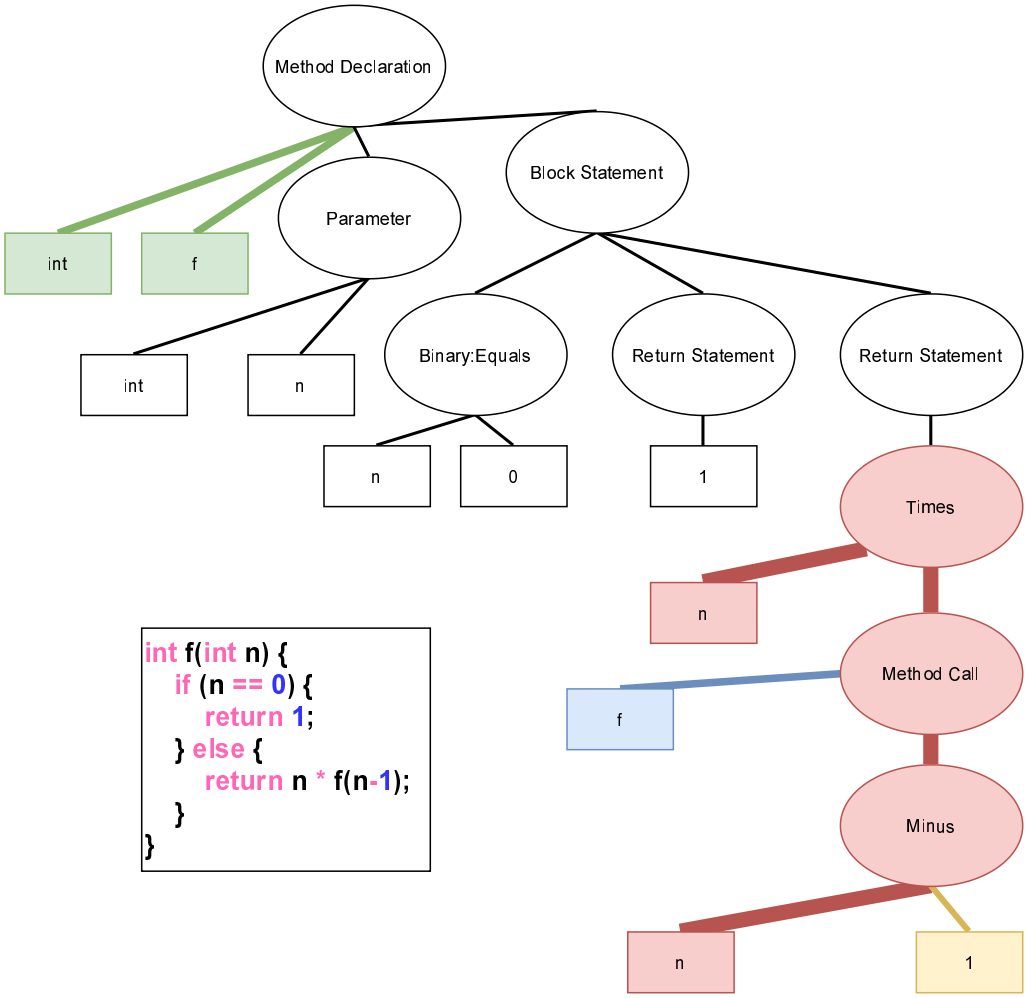

Fig. 3.1: AST representation of a factorial function

The Affinity Propagation algorithm is based on two concepts that are passing messages between data points and finding exemplars [19]. Exemplars are the centres of each cluster, which represent the cluster, and each cluster contains a single exemplar. Also, there are two types of these exchanged messages between the data points. The first type is exchanged between the data points and the candidate exemplars, and these types of messages are called (responsibility) messages. They are used to find the strength of the link between the data points and the exemplars.

The (responsibility) messages are represented by $r(i,k)$ in equation 3.1 implies if point k is fit to be an exemplar for point i. Responsibilities are exchanged from point i to exemplar to be k:

$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + s(i,k')\} \tag{3.1}$$

The second type checks the suitability of an exemplar in being an exemplar by sending messages from the exemplar candidates to other data points in the cluster. This type of messages referred to as (availability) messages. The (availability) represented by $a(i,k)$ in equation 3.2 shows if point i can select point k as an exemplar. Availabilities are exchanged between exemplar candidate k and data point i starting from k:

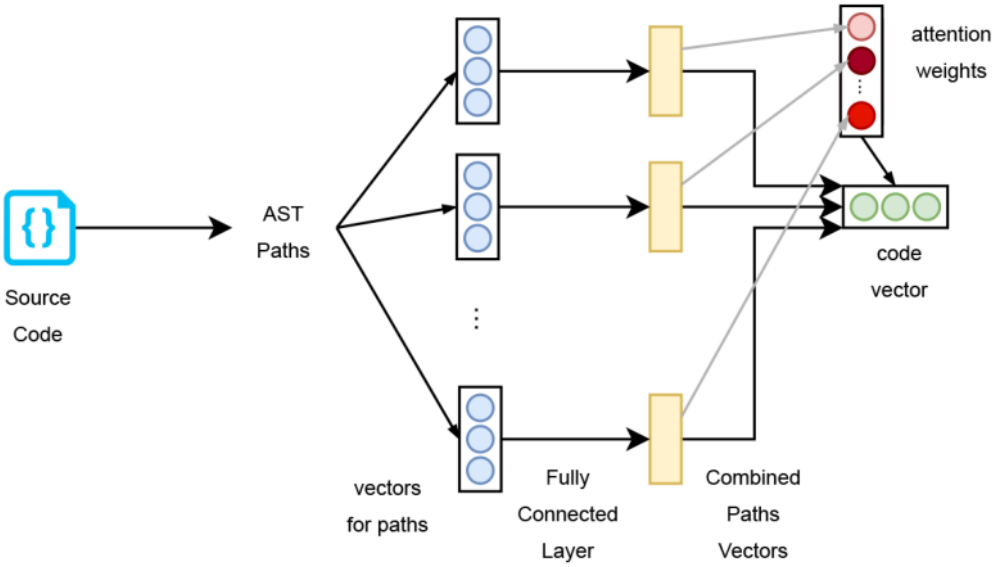

Fig. 3.2: code2vec Model [22]

Table 3.1: Aggregation Methods

Aggregation Method	Equation
Mean	$\frac{1}{n}\sum_{i=1}^{n} x_i$
Summation	$\sum_{i=1}^{n} x_i$
Maximum	$x_i : x_i \geq x_i, i \neq j \forall i, j \in n$
Minimum	$x_i : x_i \leq x_i, i \neq j \forall i, j \in n$
Standard Deviation	$\sqrt{\frac{\sum(x_i - \overline{x})^2}{(n-1)}}$
Variance	$\frac{\sum(x_i - \overline{x})^2}{(n-1)}$

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ s.t. } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\} \tag{3.2}$$

Equation 3.3 shows the method of updating self-availability for an exemplar:

$$a(k,k) \leftarrow \sum_{i' \text{ s.t. } i' \neq k} \max\{0, r(i',k)\} \tag{3.3}$$

Then pairwise similarities are used to identify the similarities between the data points. Also, clusters can be found by maximizing the total similarity between the exemplars and their data points.

Mezard [20] described the significance and effectiveness of message passing algorithms, even on complex problems. Thus, Affinity Propagation was used for our research paper for clustering related classes to generate microservices candidates.

Affinity Propagation algorithm includes three parameters which affect the performance of the clustering algorithm:

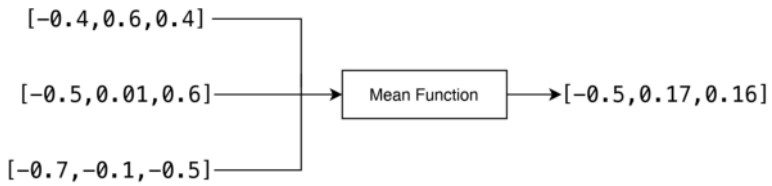

Fig. 3.3: Mean Aggregation Method

1. The first parameter is damping, which checks the interchange of messages between responsibility and availability to avoid numerical fluctuations while updating the values of responsibilities and availabilities [21].
2. The second parameter is the maximum number of iterations.
3. The third one is the number of iterations with no change in the number of estimated clusters that stop the convergence.

Algorithm 1 shows the steps of the Affinity Propagation algorithm.

Algorithm 1 Affinity Propagation algorithm

Input: $\{s(i,j)\}_{i,j\in\{1,\ldots,N\}}$ data similarities and preferences
Output: cluster assignments $\hat{c}$
Availability$\leftarrow$0
repeat a and r updates until convergence
  $r(i,k) \leftarrow s(i,k) - \max_{k' \text{s.t.} k'\neq k}\{a(i,k') + s(i,k')\}$
  $a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ s.t. } i'\notin\{i,k\}} \max\{0, r(i',k)\}\right\}$
  if $k \neq i$ **then**
    $a(k,k) \leftarrow \sum_{i' \text{ s.t. } i'\neq k} \max\{0, r(i',k)\}$
  end if
until convergence
Return $\hat{c} = argmax_k\,[a(i,k) + r(i,k)]$

Affinity Propagation groups similar code embeddings together in order to generate microservices candidates. The proposed microservices candidates are analysed using cohesion metrics in order to be compared with the results of other decomposition methods.

3.5. Metrics for Evaluating Clustering Method Performance. Silhouette coefficient, precision, recall, and f-measure were used to determine the efficiency and the threshold of the clustering method parameters. Silhouette coefficient $s(i)$ [28] is a validation method for clustered data. It measures the similarity of an object within its cluster and compares it to other clusters. An object is perfectly matched to its cluster when it gets a score of 1, and it is incorrectly matched when it gets a score of -1, so $s(i)$ score ranges from -1 to 1. The silhouette coefficient was used to evaluate the effectiveness of the clustering method with different parameters setup. The silhouette coefficient is shown in equation 3.4.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{3.4}$$

where $a(i)$ is the mean dissimilarity for object i, compared to the other objects in the same cluster. $b(i)$ is the smallest average distance between i and other data points in different clusters. Also, a grid search was utilized in order to find the most suitable values for the cluster algorithm parameters.

Besides the silhouette coefficient, precision and recall [29] were used to measure the performance of the clustering algorithm and its parameters. These metrics measure the efficiency of the information retrieval

method and how the retrieved results by the method are related to the requested data. Precision is defined as shown in equation 3.5.

$$P = \frac{TP}{TP + FP} \tag{3.5}$$

where P is precision, TP represent true positive results, and FP represents false positive results. The recall definition can be found in equation 3.6.

$$R = \frac{TP}{TP + FN} \tag{3.6}$$

where FN represents false-negative results. In order to get the harmonic mean of precision and recall, we used F-Measure ($F1$) to calculate the average of the precision and recall metrics, where 1 represents the best value, and 0 is the worst. $F1$ definition can be found in equation 3.7.

$$F1 = 2 * \frac{P * R}{P + R} \tag{3.7}$$

3.6. Evaluation Metrics. For this section, we chose metrics that were used by other researchers, as well. As a result, the comparison can be suitable with other decomposition methods. These researches [12], [16], and [14] used these metrics.

The first metric is Cohesion at Message Level (CHM) which uses the average cohesion of microservices interfaces at the message level. It is a refined version of Lack of Message Level Cohesion by Athanasopoulos et al. [18]. CHM value can be calculated, as shown in equation 3.8.

$$CHM = \frac{\sum_{j=1}^{\sum_{i=1}^{K} n_i} CHM_j}{\sum_{i=1}^{K} n_i}$$
$$\text{where} \quad CHM_j = \begin{cases} \frac{\sum_{(k,m)} fsimM(Op_k, Op_m)}{|I_i| * (|I_i|-1)/2} & if\ |I_i| \neq 1 \\ 1 & if\ |I_i| = 1 \end{cases} \tag{3.8}$$
$$fsimM(Op_k, Op_m) = \frac{(\frac{|res_k \bigcap res_m|}{|res_k \bigcup res_m|} + \frac{|pas_k \bigcap pas_m|}{|pas_k \bigcup pas_m|})}{2}$$

n_i represents the number of the interfaces of a microservice i. k represents the number of microservices candidates that were generated from the monolithic application. $CHMj$ measures the cohesion of a microservice at the message level. Op_k and Op_m represent the operations that are provided by the interface "I_i" of a microservice. The similarity between the output parameters and the input parameters are calculated by the similarity function $fsimM$. The higher value of the CHM metric is the better.

The other metric is Cohesion at Domain Level (CHD), which measures the average of the interfaces' cohesion at the domain level. It is a modified version of Lack of Domain Level Cohesion by Athanasopoulos et al. [18]. The formal definition of the metric is shown in equation 3.9.

$$CHD = \frac{\sum_{j=1}^{\sum_{i=1}^{K} n_i} CHD_j}{\sum_{i=1}^{K} n_i}$$
$$\text{where} \quad CHD_j = \begin{cases} \frac{\sum_{(k,m)} fsimD(Op_k, Op_m)}{|I_i| * (|I_i|-1)/2} & if\ |I_i| \neq 1 \\ 1 & if\ |I_i| = 1 \end{cases} \tag{3.9}$$
$$fsimD(Op_k, Op_m) = \frac{|T_{Op_k} \bigcap T_{Op_m}|}{|T_{Op_k} \bigcup T_{Op_m}|}$$

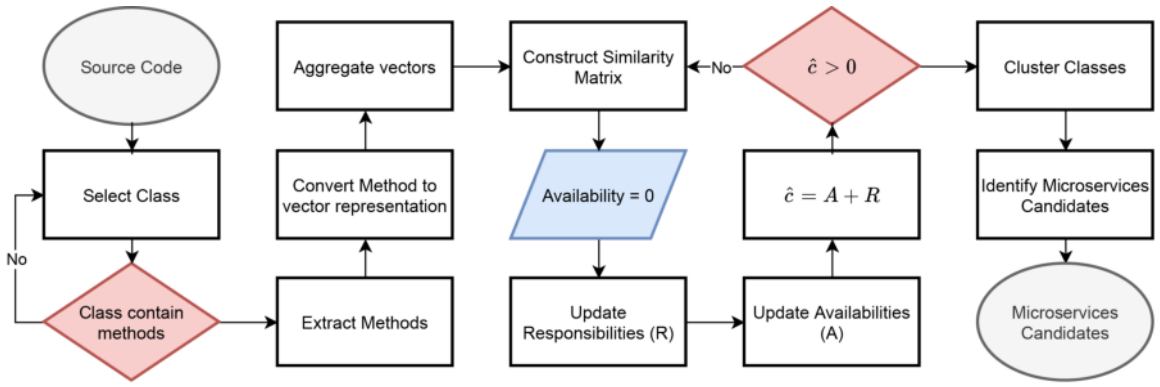

Fig. 3.4: High - Level Representation of the Proposed Algorithm

Table 4.1: Dimensions of the Tested Applications

Application	Classes	Methods	LoC	MS numbers
JPetStore	24	290	2059	4
SpringBlog	46	155	1553	6
JForum	335	2702	52,719	8
Roller	153	780	29,154	11

n_i represents the number of the interfaces of a microservice i. K represents the number of microservices candidates that were generated from the monolithic application. $fsimD$ function calculates the similarity of the operations at the domain level. Op_k and Op_m represents the domain terms that are extracted from the operations. The higher value of the CHD metric is the better.

CHM and CHD metrics were introduced by Jin et al. [12]. These metrics are used for measuring the cohesion at message and domain levels of the microservices through analysing their interfaces.

Fig. 3.4 presents a high-level description of the proposed algorithm, which starts with obtaining the methods code snippets from the monolithic application source code. Then these codes are converted to code embeddings using the code2vec model. Furthermore, aggregate the methods code embeddings using the mean function to represent the code of each class of the related methods. Finally, microservices candidates are generated through clustering related class files using a hierarchical clustering algorithm.

4. Experiments and Results. The setup of the experiment consists of testing four applications to compare the performance of the proposed method against other methods in the literature. The first application is JPetStore[1] is a pet store commercial website written in JAVA, and it is a monolithic web application consists of 24 classes. Also, it is the smallest application in the experiment setup. The second application is SpringBlog[2], which is a blogging website that is written in JAVA consists of 46 classes. The third application is JForum[3], which is a messaging boards application consisting of 335 classes. The last application is Apache Roller[4], which is a monolithic application that allows multiple users to create blog sites and posts. The sizes of these applications range from small to big applications with different class numbers, method numbers, and lines of codes. See a detailed comparison of the tested applications in Table 4.1.

[1]https://github.com/mybatis/jpetstore-6
[2]https://github.com/Raysmond/SpringBlog
[3]https://sourceforge.net/projects/jforum2/
[4]https://github.com/apache/roller

Table 4.2: Aggregation Methods Accuracy Comparison

Aggregation Method	Accuracy	Precision	Recall	F1	Silhouette coefficient
Mean	0.70	0.58	0.46	0.49	0.47
Sum	0.07	0.07	0.008	0.015	N/A
Maximum	0.15	0.33	0.10	0.14	0.27
Minimum	0.15	0.33	0.10	0.14	0.23
Median	0.23	0.56	0.27	0.30	0.17
Standard deviation	0.15	0.05	0.33	0.09	N/A
Variance	0.15	0.05	0.33	0.09	N/A

Table 4.3: JPetStore Metrics Scores

Application	Metrics	Jin et al	Our Method
JPetStore	CHD	0.52	0.52
	CHM	0.78	0.82

4.1. Aggregation Method. Several aggregation methods were tested to find the most effective method for the proposed algorithm. These methods are mean, sum, standard deviation, variance, maximum, and minimum. The setup for the experiment consisted of comparing the accuracy, precision, and recall of the clustering results against the optimal microservices design of Spring Pet Clinic[5], which have the monolithic application and the microservices design[6] as well. The results of the experiment are shown in Table 4.2. Thus, the mean function is the most suitable aggregation method for this experiment because it has the highest scores for accuracy, precision, and recall.

4.2. Clustering Method Parameters. The parameters for refining the performance of the Affinity Propagation algorithm are damping, the maximum number of iterations, and convergence iterations, the values for these parameters were 0.8, 500, and 50, respectively. These values were found using the gird search technique with different setups, configurations, and tests against the monolithic application Spring Pet Clinic which was mentioned previously. The results for these tests are displayed in Table 4.2. The tests were compared using the silhouette coefficient score.

4.3. Decomposition Results. After conducting the previous experiments and tests to find the most optimal aggregation method and the most efficient parameter values, it is the turn of displaying the results of the proposed decomposition methodology. As was mentioned before in Section 4, the decomposition method was tested with four different applications (listed in Table 4.1.)

The first application is JPetStore, which was tested by Jin et al. [12] and Saidani et al. [14]. JPetStore application was compared with Jin et al. approach in detail. For example, Fig. 4.1 shows a comparison between the decomposition results of our approach and their approach. Our approach generated four microservices while Jin et al. approach gave three microservices. Fig. 4.1 displays the microservices and their related classes.

For the cohesion side of the comparison, both of the approaches have similar results, but our approach has a slightly better score for CHM. These results in Table 4.3 are concerning the results of only JPetStore application because the decomposition results for JPetStore were described thoroughly in the research of Jin et al. [12].

The results for the comparison of the proposed method and the other methods using the additional three applications are available in Table 4.4.

[5]https://github.com/spring-projects/spring-petclinic
[6]https://github.com/spring-petclinic/spring-petclinic-microservices

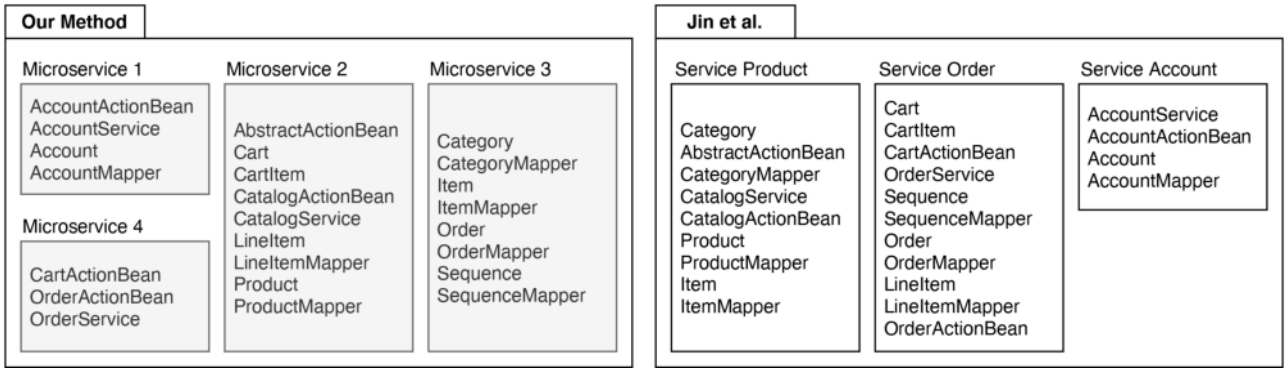

Fig. 4.1: JPetStore Results

Table 4.4: Decomposition Results

Application	Metrics	Jin et al	Saidani et al.	Our Method
JPetStore	CHD	0.52	0.65	0.52
	CHM	0.78	0.55	0.82
SpringBlog	CHD	0.55	0.67	0.50
	CHM	0.68	0.75	0.73
JForum	CHD	0.45	0.15	0.52
	CHM	0.70	0.51	0.73
Roller	CHD	0.52	0.38	0.53
	CHM	0.72	0.78	0.76

The second application is SpringBlog, which consists of 46 classes. The results in Table 4.4 suggest that our approach have a better performance in term of CHM metric compared to the other decomposition methods, but our approach has a less cohesive score, based on the CHD score, compared to the other approaches.

For the JForum application, the proposed method performed the best in terms of cohesion at the message and domain level, as it is shown in Table 4.4. It scored better scores in both CHD and CHM compared to Jin et al. and Saidani et al. methods. Therefore, this means the proposed method creates better decomposition results in terms of cohesion.

The final application is Apache Roller, where our approach had slightly improved results in term of CHD, while had a good result for CHM metrics. These results show that the proposed method can handle big applications such as JForm and Apache Roller without any issues.

The overall results for tested applications suggest that our approach has some advantages in terms of cohesion in the middle and high range applications. For example, Table 4.4 shows that most of the better and good metrics values were related to our approach, except in the small tier application such as JPetStore. Our approach scored the best results in five test experiments out of 8, while Saidani et al. method scored 3 out of 8, and Jin et al. scored 0. These results show that all the methods have good results, but the proposed method had better ones when compared with the other methods. The proposed method showed better performance in terms of cohesion, which is one of the essential requirements for a good microservices application design because microservices applications need to be loosely coupled and cohesive, according to Newman [4].

Fig. 4.2 shows an interpretation of the results that are shown in Table 4.4. Fig. 4.2 shows that our method is performing similar to Jin et al. [12] but in 4 cases has better performance. Also, the results of Saidani et al. [14] fluctuates between 0.1 and 0.8, while the results of the proposed method are between 0.5 and 0.8. Therefore, this means the proposed method has a more stable approach when compared to Saidani et al. approach.

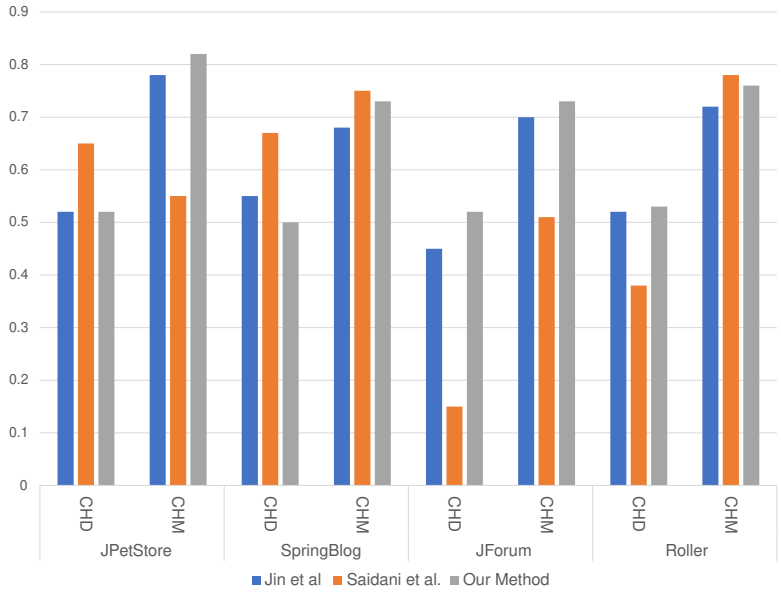

Fig. 4.2: Metrics Results Comparing the Performance of the Decomposition Methods

Table 4.5: Average Results of CHD and CHM Metrics

Metrics	Jin et al	Saidani et al.	Our Method
CHD Average	0.51	0.46	0.52
CHM Average	0.72	0.65	0.76

The overall results showed that the proposed decomposition method is better performing compared to Jin et al. and Saidani et al. methods. For example, our method had better results in 5 out of 8 metrics scores, Saidani et al. had 3, and Jin et al. 's method performed the worst when compared to the other methods. In another interpretation of the results, Table 4.5 presents the averaged results of Table 4.4, which shows that the results of Jin et al. are better on average compared to Saidani et al., but our proposed method has the best results in this case as well.

5. Conclusion. This paper proposed a novel decomposition method for refactoring monolithic applications into microservices applications using a neural network based model for creating code embeddings from the monolithic application source code. As a Result, semantically similar code embeddings are grouped using a hierarchical clustering algorithm in order to generate microservices candidates. The quality characteristics of the results were measured using two metrics for measuring cohesion.

The proposed method showed promising results in terms of cohesion when compared to other decomposition methods. The results were compared with two other methods proposed by Jin et al. [12] and Saidani et al. [14], 8 test cases were conducted, and the proposed method got the highest scores in 5 of them.

In conclusion, the proposed method can be a helpful add-on for developers in the process of migration from monolithic architecture into microservices architecture. This method will give the developers insights and directions on the path and the design that the developers need to take in order to achieve a good microservices design.

For future work, this method can be developed further and can be tested with other programming languages such as Python, C, C++, et al. The tested cases of this research were all written in JAVA, and the proposed method is only capable of handling code written in that programming language. Also, the neural network-based

model can be trained on the source codes of the microservices application to achieve more precise results.

REFERENCES

[1] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Software 33(3):42–52, 2016.

[2] Cerny, Tomas and Donahoo, Michael J. and Trnka, Michal. Contextual Understanding of Microservice Architecture: Current and Future Directions. ACM SIGAPP Applied Computing Review 17(4):29–45, 2018.

[3] Dragoni, Nicola and Giallorenzo, Saverio and Lafuente, Alberto Lluch and Mazzara, Manuel and Montesi, Fabrizio and Mustafin, Ruslan and Safina, Larisa. Microservices: yesterday, today, and tomorrow In: Mazzara M., Meyer B. (eds) Present and Ulterior Software Engineering. Springer, Cham, 2017.

[4] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Incorporated. 2021.

[5] Nuno Santos and António Rito Silva. A complexity metric for microservices architecture migration. In *2020 IEEE International Conference on Software Architecture (ICSA)*, pages 169–178, 2020.

[6] Shanshan Li, He Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, Jidong Ge, and Zhihao Shan. A dataflow-driven approach to identifying microservices from monolithic applications. Journal of Systems and Software 157, 2019.

[7] Omar Al-Debagy and Peter Martinek. A new decomposition method for designing microservices. Periodica Polytechnica Electrical Engineering and Computer Science 63(4):274–281. 2019.

[8] Manabu Kamimura, Keisuke Yano, Tomomi Hatano, and Akihiko Matsuo. Extracting candidates of microservices from monolithic application code. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 571–580, 2018.

[9] Genc Mazlami, Jürgen Cito, and Philipp Leitner. Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531, 2017.

[10] Luís Nunes, Nuno Santos, and António Rito Silva. From a monolith to a microservices architecture: An approach based on transactional contexts. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11681 LNCS, pages 37–52. Springer Verlag, 2019.

[11] Davide Taibi and Kari 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*, pages 153–164. SCITEPRESS - Science and Technology Publications, 2019.

[12] Wuxia Jin, Ting Liu, Qinghua Zheng, Di Cui, and Yuanfang Cai. Functionality-oriented microservice extraction based on execution trace clustering. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 211–218, 2018.

[13] Luciano Baresi, Martin Garriga, and Alan De Renzis. Microservices identification through interface analysis. In Flavio De Paoli, Stefan Schulte, and Einar Broch Johnsen, editors, *Service-Oriented and Cloud Computing*, volume 10465, pages 19–33. Springer International Publishing, 2017.

[14] Islem Saidani, Ali Ouni, Mohamed Wiem Mkaouer, and Aymen Saied. Towards automated microservices extraction using muti-objective evolutionary search. In Sami Yangui, Ismael Bouassida Rodriguez, Khalil Drira, and Zahir Tari, editors, *Service-Oriented Computing*, volume 11895, pages 58–63. Springer International Publishing, 2019.

[15] Muhammad Abdullah, Waheed Iqbal, and Abdelkarim Erradi. Unsupervised learning approach for web application auto-decomposition into microservices. Journal of Systems and Software 151:243–257, 2019.

[16] Wuxia Jin, Ting Liu, Yuanfang Cai, Rick Kazman, Ran Mo, and Qinghua Zheng. Service candidate identification from monolithic systems based on execution traces. IEEE Transactions on Software Engineering, 2019.

[17] Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. Service cutter: A systematic approach to service decomposition. In Marco Aiello, Einar Broch Johnsen, Schahram Dustdar, and Ilche Georgievski, editors, *Service-Oriented and Cloud Computing*, volume 9846, pages 185–200. Springer International Publishing, 2016.

[18] Dionysis Athanasopoulos, Apostolos V. Zarras, George Miskos, Valerie Issarny, and Panos Vassiliadis. Cohesion-driven decomposition of service interfaces without access to source code. IEEE Transactions on Services Computing 8(4):550–562, 2015.

[19] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. Science 315(5814): 972–976, 2007.

[20] Marc Mézard. Computer science. Where are the exemplars? Science 315(5814): 949–951, 2007.

[21] R. Refianti, A. B. Mutiara, and A. A. Syamsudduha. Performance evaluation of affinity propagation approaches on data clustering. International Journal of Advanced Computer Science and Applications 7(3), 2016

[22] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. code2vec: learning distributed representations of code. Proceedings of the ACM on Programming Languages 3:1–29, 2019.

[23] Brahmaleen Kaur Sidhu, Kawaljeet Singh, and Neeraj Sharma. A machine learning approach to software model refactoring. International Journal of Computers and Applications 0(0):1–12, 2020

[24] Boukhdhir Amal, Marouane Kessentini, Slim Bechikh, Josselin Dea, and Lamjed Ben Said. On the use of machine learning and search-based software engineering for ill-defined fitness function: A case study on software refactoring. In Claire Le Goues and Shin Yoo, editors, *Search-Based Software Engineering*, Lecture Notes in Computer Science, pages 31–45. Springer International Publishing, 2014.

[25] Yasemin Kosker, Burak Turhan, and Ayse Bener. An expert system for determining candidate software classes for refactoring. Expert Systems with Applications 36(6): 10000–10003, 2009.

[26] Rhys Compton, Eibe Frank, Panos Patros, and Abigail Koay. Embedding java classes with code2vec: Improvements from variable obfuscation. In Proceedings of the 17th International Conference on Mining Software Repositories, 243-253, 2020.

[27] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. A general path-based representation for predicting program properties. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 404–419, 2018.

[28] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics 20:53–65, 1987.

[29] Kai Ming Ting. Precision and recall. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 781–781. Springer US, 2010.

Edited by: Dana Petcu
Received: Nov 18, 2020
Accepted: Jan 21, 2021