IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 4 February 2024, accepted 22 March 2024, date of publication 16 April 2024, date of current version 24 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3389700

RESEARCH ARTICLE

Enhancing Automated Microservice Decomposition via Multi-Objective Optimization

TAKAHIRO KINOSHITA AND HIDEYUKI KANUKA
Research and Development Group, Hitachi Ltd., Totsuka-ku, Yokohama-shi, Kanagawa 244-0817, Japan
Corresponding author: Takahiro Kinoshita (takahiro.kinoshita.jr@hitachi.com)

ABSTRACT The microservices architecture (MSA) has become widespread across various industries to enhance the maintainability of applications. However, manual migration of monolithic applications to MSAs via microservice decomposition (MSD) can be intricate and may adversely impact the overall maintainability of a system if not executed correctly. To address these challenges, a multi-objective optimization approach can be used to generate optimal solutions, known as Pareto-optimal solutions. However, selecting the optimal MSD solution from the set of Pareto-optimal solutions can be challenging. To mitigate this challenge, we propose a multi-objective MSD method of using reference lines, a mathematical concept used in multi-objective optimization approaches, to efficiently select the best MSD solutions. We also define a set of violations and fix operations on the basis of MSD policies to prevent generating a vast amount of semantically meaningless MSD solutions. The definition of violations and fix operations and the use of reference lines accelerates the generation of MSD solutions. Our proposed method aids information technology architects by streamlining hyperparameter determination, a task deemed intricate for such architects.

INDEX TERMS Microservice, decomposition, multi-object optimization.

I. INTRODUCTION

The microservices architecture (MSA) is becoming widespread in various industries as a means to enhance application maintainability [1]. Under an MSA, applications are created by building small services known as microservices, each of which contains its own business logic and data and executes a specific function. Microservices interact with one another through interfaces such as application programing interfaces (APIs). By embracing an MSA, microservices can be analyzed, comprehended, and updated without the need for knowledge about the internal workings of other microservices [2]. This helps localize the impact of any application changes, thereby improving software maintainability.

Many industries are now transitioning from a monolithic application to an MSA application to enhance the maintainability of their applications. Monolithic application embody multiple logics, data, and functions, making the process of updating a singular function challenging. It is noteworthy that many industries continue to operate with large-scale and intricate monolithic application.

Manual migration from a monolithic application to an MSA is a complex task for information technology (IT) architects. For example, the granularity inherent in microservices necessitates a delicate balance between program structure and business requirements. If one overly emphasizes the business requirements, the resulting microservice may become too fine-grained. While this might initially seem beneficial, it can lead to challenges such as database (DB) consistency issues or complications arising from frequent data-structure changes. Conversely, if the focus leans too heavily on program structure, it may hinder the software's ability to evolve and adapt to changing business needs in a timely manner. Achieving the right equilibrium between these factors is crucial for the successful implementation and long-term viability of a microservice-based system. Improper MSD for

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

a project by overlooking the analysis of program structure or business requirement can also compromise the overall maintainability of the software application [3], [4] which is metaphorically labeled as a ''distributed monolith''.

Therefore, it is imperative to conduct a thorough trade-off analysis to determine the most suitable microservices for an application [5]. This requires comparing various MSD solutions to comprehensively analyze the associated trade-offs.

One solution to the daunting MSD task is to use a multi-objective optimization approach, which produces a set of optimal solutions known as Pareto-optimal solutions by simultaneously considering multiple objectives. This approach is particularly useful for tackling intricate problems that involve conflicting objectives, as it can offer a range of trade-offs between them [6]. A multi-objective approach can aid in identifying the optimal balance among various objectives, resulting in a collection of Pareto-optimal solutions that offer distinct trade-offs.

The multi-objective approach, however, may cause issues such as generation of a vast amount of semantically meaningless MSD solutions and selection of a solution from a large amount of Pareto-optimal solutions. A simple method for generating a new solution for MSD is to choose one program function randomly and move it to another microservice [7], which may generate a large amount of solutions that have less semantic meaning for IT architects. Generating solutions on the basis of semantic factors that IT architects consider suitable is required to support such architects in executing MSD [6], [8]. IT architects also struggle to determine the most appropriate solution for their project due to the large amount of Pareto-optimal solutions generated. To tackle this issue, Zhang et al. [9] focused on a specific objective and ordered the solutions on the basis of their performance on this objective. Mkaouer et al. [7] selected the solution located at the Knee Point [10] (i.e., the tip point of a convex part) from the set of Pareto-optimal solutions. However, such an approach may not be sufficient, as the priority of each objective varies depending on the project requirements.

To address the issue of the multi-objective approach, we propose a multi-objective MSD method for generating MSD candidates (MSDCs) on the basis of MSD policy and selecting MSDCs using reference line. We handled the issues of generating meaningful MSDCs by predefining a set of MSD policies and corresponding violations and fix operations. These policies and operations are designed to ensure that the generated MSDCs are semantically meaningful and compliant with the design principles and best practices of an MSA. We also address the issues of selecting MSDCs on the basis of a reference line which is a mathematical concept used with multi-objective optimization method NSGA-III [11]. NSGA-III uses a reference line to determine the direction of the solutions on the basis of their proximity to the reference line. The perpendicular distance is used to calculate the distance between a solution and reference line, and solutions closer to the reference line are likely to remain. Our method aligns the reference-line vector evenly in the objective space, and selects the Pareto-optimal MSDCs that are closer to the reference line.

We have formulated the following research questions:

RQ1: What is the impact of defining an MSD policy on the generation of Pareto-optimal MSDCs?

RQ2: How does the use of reference lines in the generation of MSDCs contribute to our proposed method and what role do the component values of the reference-line vector play?

RQ3: How does our method provide support for IT architects compared with previous methods?

Our contributions to the field can be summarized as follows:

- The generation of new MSDCs on the basis of MSD policy decrease the number of MSDCs with lower semantic meaning, limiting the quantity of Pareto-optimal MSDCs. However, the generation time of new MSDC are likely to increase depending on the number of program and data structures that violate MSD policies.
- The use of reference lines accelerates the generation of MSDCs by providing a framework for efficient choice and refinement of MSDC. The component values of the reference-line vector serve as indicators of the distinctive features of the selected MSDCs. This enables a comparative analysis of the advantages and disadvantages of each MSD solution, facilitating fine-tuning and decision-making processes.
- Our method aids IT architects in navigating uncertainties related to the hyperparameter that determines the number of partitions within a monolithic system, which is essential for its transformation into microservices—a requirement highlighted in prior studies to generate MSDCs automatically.

In Section II, we discuss previous research in the field. In Section III, we explain our method and how it handles challenges from previous studies. Section IV discusses the data we collected from applying our method to three software applications and present the findings. In Section V, we explore how our method generates various solutions, selects those with different features, and aids IT architects in decision-making, on the basis of our findings. In Section VI, we outline our method's validity. Finally, we present our conclusions in Section VII.

II. RELATED WORKS

There are two main approaches to MSD: manual and automatic. In the manual approach, the IT architect carries out the task either by hand or with tool assistance. The automatic approach, on the other hand, involves minimal to no intervention from the IT architect. This approach can be further divided into two approaches: clustering and search-based. Both of these approaches address a graph partitioning problem, which is known to be NP-hard, by seeking a near-optimal solution within a reasonable amount of time.

A. MANUAL APPROACH

Several studies executed MSD manually in various applications such as banking applications and mobile applications [12], [13]. To enhance the efficiency of manual MSD, microservice design patterns, such as *decompose by business capability* and *decompose by subdomain* [14], and semi-automatic methods have been proposed [15], [16]. Ntendos et al. [15] presented a method for automatically identifying microservices that violate the MSA pattern as a violation and presenting options for violation correction to the IT architect.

B. CLUSTERING APPROACH

In this approach, an objective such as the semantic similarity of each function is used to generate an MSD solution [17]. Even if multiple objectives are used, the MSD solution is generated by combining them into a single objective by using weighted sum methods and clustering methods such as K-means ([18], [19], [20], [21], [22], [23], [24], etc.). Gysel et al. [18] used a weighted sum method to calculate each objective and decides whether to group elements such as methods and data into the same microservice unit. Other studies [20], [21], [23] grouped elements into microservices on the basis of a predefined number of microservices determined by the IT architect.

In this approach, weighting each objective or determining the number of clusters cannot be done automatically, which remains an open issue [25]. Since this approach does not create alternative solutions for comparison, the IT architect must verify the appropriateness of the weighting or number of clusters by generating solutions and adjusting them through trial and error until the desired solution is achieved [26].

C. SEARCH-BASED APPROACH

Search-based approach for MSD can be divided into two types of methods: single-objective optimization and multi-objective optimization.

Single-objective optimization methods are used to find an optimal solution on the basis of a single objective by modifying certain parts of the existing solutions. Single-objective optimization methods, such as those using genetic algorithms and hill-climbing algorithms, include those by Doval et al. [27] and Mitchell and Mancoridis [28]. The method by Mitchell and Mancoridis [28] generates new solutions, for example, by moving one program function to another microservice in the existing solution. However, finding the optimal solution for a project remains a challenge with single-objective optimization methods, as there are multiple metrics to consider and combining them into a single objective may result in the generation of suboptimal solutions [29].

Multi-objective optimization methods are used to identify a set of solutions that are Pareto-optimal on the basis of a multiple objective by modifying certain parts of the existing solutions. Multi-objective optimization methods are said to produce better solutions than clustering approaches which optimize single-objective [29]. Various studies (e.g., [7], [9], [30], [31], [32], [33], [34], [35], [36]) used multi-objective optimization methods such as MOEA-D [37], NSGA-II [38], and NSGA-III [11]. Mkaouer et al. [7] generated new solutions by applying refactoring operations such as moving classes between groups and merging groups to the existing solutions. A predetermined number of unwanted solutions were then eliminated using non-dominated sorting [39] and reference-line vectors [40].

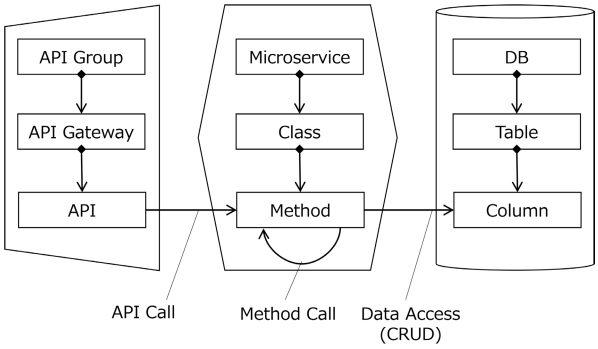


FIGURE 1. Metamodel of MSDC.

III. PROPOSED METHOD

We propose an MSD method by enhancing multi-objective optimization.

A. BASIC DEFINITION

1) METAMODEL OF MSDC

In this section, we present a metamodel of the MSDC shown in Figure 1. This MSDC consists of twelve fundamental elements, API Group, API Gateway, API, microservice, class, method, DB, table, column, API call, method call, and data access (Create, Read, Update, and Delete (CRUD)). Each API Group is associated with one or more API Groups, and each API Gateway is associated with one or more API. Likewise, each DB is associated with one or more tables, and each table is associated with one or more columns. Similarly, each microservice is associated with one or more classes, and each class is associated with one or more methods. The API call is executed between an API and method, while the method call is executed between methods. Data access (CRUD) is carried out between a method and column. We define a transaction as a collection of methods, columns, and data-access operations (CRUD) that are executed in a sequence of events during an API call.

2) MSD POLICIES

With our method, we have formulated four MSD policies (P1–P4) to identify Pareto-optimal MDSCs. These policies are defined on the basis of previous studies on microservices' application maintainability [14], [41].

a: P1. ONE BUSINESS CAPABILITY PER MICROSERVICE

This policy assesses whether each microservice comprises only one pre-defined business functions as specified by the IT architect. The IT architect defines business functions by identifying the business function unit and associating the API Gateway, class and the DB table of the monolithic application to each business function unit. The concept of a business function is subjective and cannot be determined solely by analyzing the source-code structure, such as cohesion and coupling [8].

b: P2. PRESPECIFIED METHOD CALL ONLY

This policy evaluates whether there are any method calls between microservices that are not pre-specified by the IT architect. The method calls between microservices can be converted to API calls. However, not all method calls between microservices are appropriate, as method calls between microservices that are not expected by the IT architect may lead to unexpected errors when modifying microservices.

c: P3. WRITE ACCESS FROM ONE MICROSERVICE PER DB TABLE

This policy measures whether each DB table can be mapped to one microservice. This policy is related to the *decompose by subdomain* pattern. Each DB table represents a domain model of an application, and each microservice owns a domain model. Write access (Create, Update, Delete (CUD)) of the microservice to the DB table can be determined as the ownership of the DB table, hence the domain model for the microservice.

d: P4. NON-DISTRIBUTED TRANSACTIONS

This policy is related to code readability in an MSA. As MSD may break up some of the current transactions, patterns such as Try-Commit-Cancel (TCC) and Saga must be incorporated into the code, which increases the code unrelated to business logic and reduces code readability.

B. MSD OPTIMIZATION MODEL

To modelize MSD optimization using MSD policies, our method introduces the concepts of violations and evaluation functions for each policy. The violations and evaluation functions for each policy are listed in Table 1. Violations V1–V4 are formulated to detect program and data structure components that violate MSD policies P1–P4.

The goal with this MSD optimization problem is to find the set of Pareto-optimal MSDCs that minimize each evaluation function value. The problem is formulated as finding the MSDC x that minimizes the evaluation function $EF_n(x)$ for $n = 1, 2, \ldots, N$, where N represents the number of evaluation functions. The number of evaluation function depends on the number of MSD policies.

$$\arg\min_{x \in \text{MSDCs}} EF_n(x), n = 1, 2, \ldots, N$$

C. PROPOSED METHOD

An overview of our method is illustrated in Figure 2.

Our method relies on using an initial MSDC and violation-state exception list, both assumed to be created by the IT architect. The initial MSDC is formulated through the partitioning of the monolithic application into microservices on the basis of business functions. This initial MSDC is stored and used to create a new MSDC, subsequently establishing MSD policy P1. The violation-state exception list is created by pre-specifying acceptable method calls between microservices and is used to prevent undesired refactoring.

After the input of initial MSDC and violation-state exception list, the processes A-1, A-2, and A-3 are iterated for a predetermined number (Step A in Figure 2). In process A-1, MSDCs are extracted from the data storage. For each MSDC, program structures that violate the MSD policies are detected as violations. In process A-2, each detected violation is treated with the relevant fix operation to generate new MSDCs. In process A-3, Pareto-optimal MSDCs are retained from the generated set of MSDCs, while the remaining ones are discarded. We note that the number of iterations should exceed that of violations in the initial MSDC to ensure comprehensive coverage in addressing all violations.

After the iterations, a number of MSDCs is selected from the generated MSDCs (Step B in Figure 2), with the number of selection assumed to be predetermined by the IT architect. An MSDC explanation document is also generated as an output to illustrate the characteristics of each MSDC. The selection of MSDCs and the generation of the MSDC explanation document are achieved using reference-line vectors [40].

We now describe the details of Steps A and B.

1) STEP A: GENERATION OF PARETO-OPTIMAL MSDCS

a: A-1. VIOLATION DETECTION FOR EACH MSDC

First, all the MSDCs are extracted from the designated data storage. Subsequently, for each MSDC extracted, all violations (V1–V4 in Table 1) are identified by examining the program and data structure.

The MSDC positioned on the left in Figure 3 illustrates an example of an MSDC with V3. Method b11 of microservice B has data write access to DB table AA1 of microservice A, which matches the condition of V3. The violation is automatically detected by analyzing the program and data structure of each MSDC.

b: A-2. FIX OPERATION APPLICATION FOR EACH MSDC

Each detected violation is addressed by applying the relevant fix operation (V1F1–V4F2 in Table 2) to generate new MSDCs. V1F1–V4F2 are intended to modify V1–V4 so that they conform to P1–P4. For instance, using V2F1, V2F2, and V2F3 will generate three new MSDCs for resolving V2.

The MSDCs positioned on the right in Figure 3 illustrates an example of MSDCs to which a fix operation is applied.

TABLE 1. MSD policies, violations, and evaluation functions.

MSD policy	Violation	Evaluation Function ($x : MSDC$)
P1. One business capability per microservice.	V1. Microservice that consists of several business functions.	$EF_1(x)$. Ratio of V1 programs and data elements in x to all the programs and data elements in x.
P2. Prespecified microservice call only.	V2. Method call between microservices that is not included in violation-state exception list.	$EF_2(x)$. Ratio of the V2 method call elements in x to all method call elements in x.
P3. Write Access from One Microservice per DB Table.	V3.Data write access that access to other microservice's DB table.	$EF_3(x)$. Ratio of V3 data write access elements in x to all data write access elements in x.
P4. Non-distributed Transactions.	V4. Transaction that has data write access in several microservices.	$EF_4(x)$. Ratio of V4 transaction elements in x to all transaction elements in x.

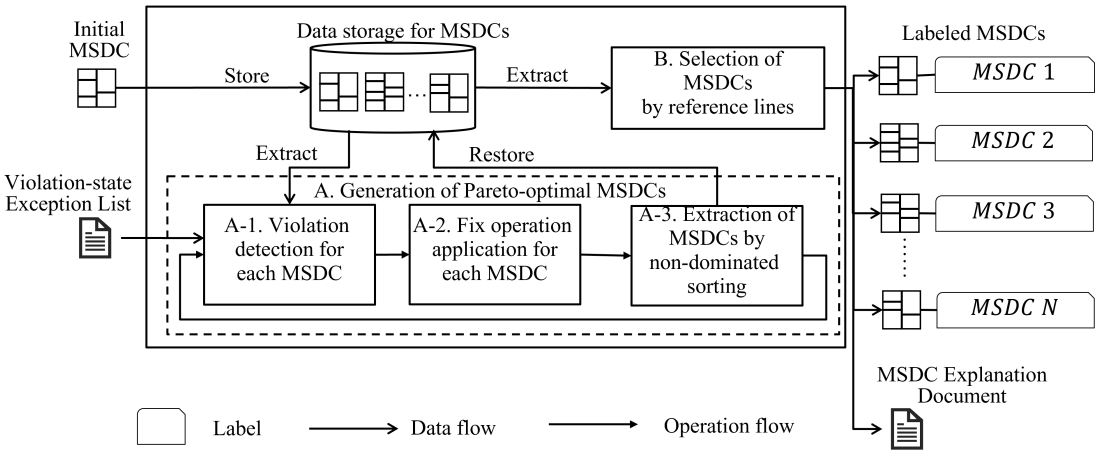


FIGURE 2. Overview of proposed MSD method.

TABLE 2. Fix operations of each violation.

Violation	Fix operation
V1	V1F1. Decompose microservice by business function.
V2	V2F1. Merge microservices.
	V2F2. Create new microservice that includes all methods having dependency on caller and callee methods.
	V2F3. Create new microservice that includes callee method.
V3	V3F1. Merge all microservices that have data write access to table.
	V3F2. Split DB table into several microservices.
	V3F3. Create new microservice that includes DB table and all methods that have data write access to the DB table.
V4	V4F1. Merge all microservices called in transaction.
	V4F2. Create new microservice including all methods and all DB tables having data write access in transaction.

New MSDCs are generated by merging microservices A and B (V3F1), splitting DB table AA1 into microservices A and B (V3F2) and creating new microservice AA1 which includes DB table AA1 and method a12, b11. By applying the applicable fix operations to the violation, several MSDCs that resolve those violations are automatically generated.

c: A-3. EXTRACTION OF MSDCS BY NON-DOMINATED SORTING

Pareto-optimal MSDCs are retrieved and stored in the designated data-storage area. A non-dominated sorting algorithm is used to obtain Pareto-optimal MSDCs from the generated MSDCs. Generated MSDCs are mapped to a vector space with $EF_1(x)$-axis, $EF_2(x)$-axis, $EF_3(x)$-axis and $EF_4(x)$-axis. The non-dominated sorting algorithm arranges MSDCs that are not dominated by other MSDCs as Rank 1 and those that are dominated by MSDCs of Rank N (N=1, 2, 3 . . .) as Rank N+1. MSDC X dominates MSDC Y if X yields better results than Y for all the evaluation functions.

Algorithm 1 illustrates the non-dominated sorting algorithm for selecting Pareto-optimal MSDCs from all the generated MSDCs. *MSDCs* denotes the set of generated MSDCs.

2) STEP B: SELECTION OF MSDCS BY REFERENCE LINES

To further refine the set of MSDCs, we use a five-step process for selecting representative MSDCs while preserving their variability.

1) Extract all Pareto-optimal MSDCs from the data storage.
2) Map the extracted MSDCs to a vector space with $EF_1(x)$-axis, $EF_2(x)$-axis, $EF_3(x)$-axis, and $EF_4(x)$-axis.

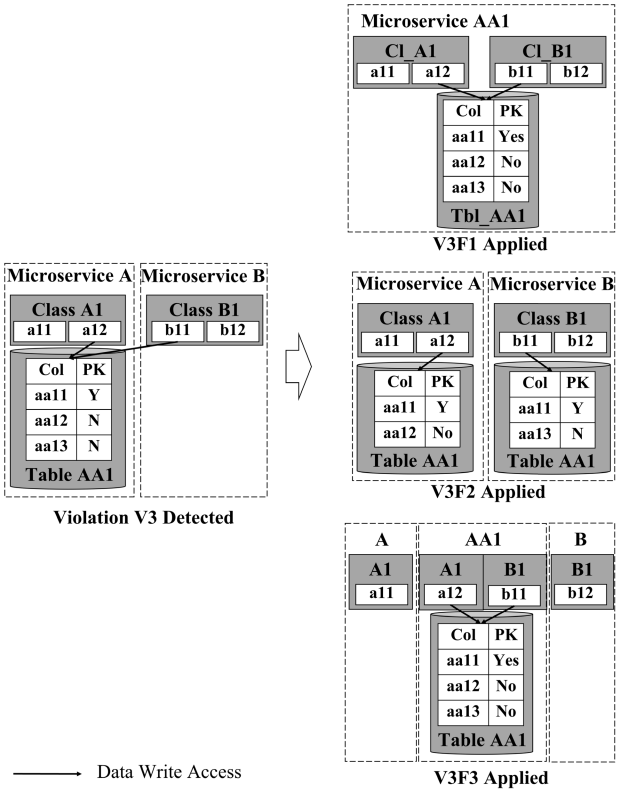


FIGURE 3. Example of applying fix operation to V3.

3) Align reference-line vectors to the vector space.
4) Extract MSDCs that are closest to each reference-line vector.
5) Label each selected MSDC to distinguish them from other MSDCs.

Our method uses reference lines to efficiently select the predetermined number of MSDCs from the set of Pareto-optimal MSDCs. The reference line is a mathematical concept used with multi-objective optimization methods such as NSGA-III [11] to determine the direction of the solutions on the basis of their proximity to the reference line. Our method aligns the reference line evenly in the vector space and selects the Pareto-optimal MSDCs that are closer to the reference line.

To align the reference lines, we use the algorithm proposed by Das and Dennis [40], as outlined in Algorithm 2. The dimension of the reference-line vector is represented as d, set to 4 with our method. The division number, denoted as p, is determined on the basis of the number of MSDCs to select that is predefined by IT architects. The algorithm yields an array of reference-line vectors, with elements arranged in the sequence of $EF_1(x)$, $EF_2(x)$, $EF_3(x)$, and $EF_4(x)$.

For a clearer understanding of the process involved in selecting MSDCs using reference lines, we illustrate this concept using the example shown in Figure 4. This figure illustrates a vector space with $EF_1(x)$-axis and $EF_2(x)$-axis. Pareto-optimal MSDCs are mapped while reference lines are

Algorithm 1 Selecting Pareto Optimal MSDCs

```
procedure ParetoOptimal(MSDCs, N = 4)
    rank1_MSDCs = {}
    for each msdc_a ∈ MSDCs do
        pareto_optimal = true
        for each msdc_b ∈ MSDCs − msdc_a do
            dominated = true
            for n = 1..N do
                if EFn(msdc_a) < EFn(msdc_b) then
                    dominated = false
                    break
                end if
            end for
            if dominated then
                pareto_optimal = false
                break
            end if
        end for
        if pareto_optimal then
            rank1_MSDCs∪ = msdc_a
        end if
    end for
    return rank1_MSDCs
end procedure
```

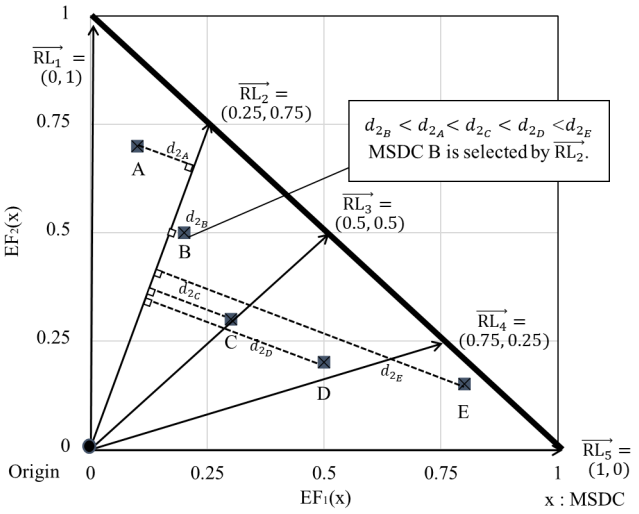


FIGURE 4. Reference line on vector space ($EF_1(x)$, $EF_2(x)$) when $p = 4$.

aligned in the vector space. In this example, the division number is set to 4 ($p = 4$), resulting in the alignment of $\overrightarrow{RL_1} - \overrightarrow{RL_5}$. Each reference line is expressed as a vector in component form relative to $EF_1(x)$ and $EF_2(x)$. When selecting MSDCs using reference lines, MSDC B, which is closest to $\overrightarrow{RL_2} = (0.25, 0.75)$ in distance, will be selected by $\overrightarrow{RL_2}$.

The reference-line vector is also used for generating the MSDC explanation document and used as a criterion for the IT architect to choose an MSDC. Figure 5 illustrates an

TABLE 3. Extraction of initial MSDC from subject application.

Element	Pet Clinic	Insurance Contract Management	Internet Banking
API Groups	Use cases of pet clinic application.	Business functions defined in specification document.	Business functions defined in specification document.
API Gateway	@Controller annotation classes.	@RestController annotation classes.	Classes that contain "Action" in their class name (action class).
API	Methods in @Controller annotation class.	Methods in @RestController annotation class.	"execute" method of action class.
Microservice	Use cases of pet clinic application.	Business functions defined in the specification document.	Business functions defined in specification document.
Class	@Service annotation classes.	@Service annotation classes.	Classes that contain "Blogic" in their class name (business logic class).
Method	Methods in @Service annotation class.	Methods in @Service Annotation class.	Methods in business logic class.
DB	Use cases of pet clinic application.	Business functions defined in specification document.	Business functions defined in specification document.
DB Table	@Entity annotation classes.	Postgres DB tables	MySQL DB tables.
Column	Global variables of @Entity annotation classes.	Columns of Postgres DB table.	Columns of MySQL DB table.
API Call	Method call between @Controller annotation class and @Service controller class.	Method call between @RestController annotation class and @Service controller class.	Method call between action class and business logic class.
Method Call	None	Method call between @Service annotation classes.	Method call between business logic classes.
Data Access	Method call between @Service annotation class and @Repository annotation class	Method call between @Service annotation class and @Repository annotation class	Method call between business logic class and data-access-object class.

TABLE 4. State of initial MSDC.

Subject	Microservices of Initial MSDC	Violation-state Exception	Number of Methods	Number of DB tables	Number of Violation
Pet Clinic	Pet, Owner, Welcome, Crash, Vet, Visit	None	24	7	3 (V1:0, V2:3, V3:0, V4:0)
Internet Banking	Common, Login, Transfer	Method calls to Common.	94	36	28 (V1:0, V2:4, V3:1, V4:23)
Insurance Contract Management	New-Contract, Billing, Maintenance, Product, Contract, Customer	Method calls to Product, Contract, and Customer.	185	36	13 (V1:0, V2:2, V3:2, V4:9)

MSDC Explanation Document

MSDC Label	EF1	EF2	EF3	EF4
MSDC 1	1	0	0	0
MSDC 2	0	1	0	0
MSDC 3	0	0	1	0
:	:	:	:	:
MSDC N	0.5	0.5	0	0

FIGURE 5. Overview of MSDC explanation document.

example of an MSDC explanation document. This document indicates that MSDC 1, MSDC 2, MSDC 3, and MSDC N were selected using the reference-line vectors (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), and (0.5, 0.5, 0, 0), respectively.

D. USE OF PROPOSED METHOD BY IT ARCHITECT

Figure 6 illustrates a flowchart of MSD by an IT architect using our method. The IT architect first splits the monolithic application into microservices on the basis of business functions as the initial MSDC. The architect also assigns the method calls to avoid refactoring to a violation-state exception list. The architect then sets p in Algorithm 2 on the basis of the number of MSDCs to select and executes our automated MSD method. Finally, the architect chooses one MSDC from the selected MSDCs on the basis on MSDC explanation document and refines it as the MSD result.

IV. EXPERIMENT AND RESULTS

In this section, we describe the experiment and the results involving our MSD method.

A. SUBJECT APPLICATION

We conducted an experiment using three software applications: pet clinic (lines of code (LOC): approximately 1000), insurance contract management (LOC: approximately 20000), and internet banking (LOC: approximately 10000). The pet clinic application is a Spring Boot application that enables scheduling appointments with a veterinarian. The insurance contract management application is a Spring MVC application developed for a life insurance company, which manages information about insurance products, customers, contracts, and operations such as new contract registration, contract maintenance, and billing. This application uses a Postgres DB, and the program accesses the DB table using

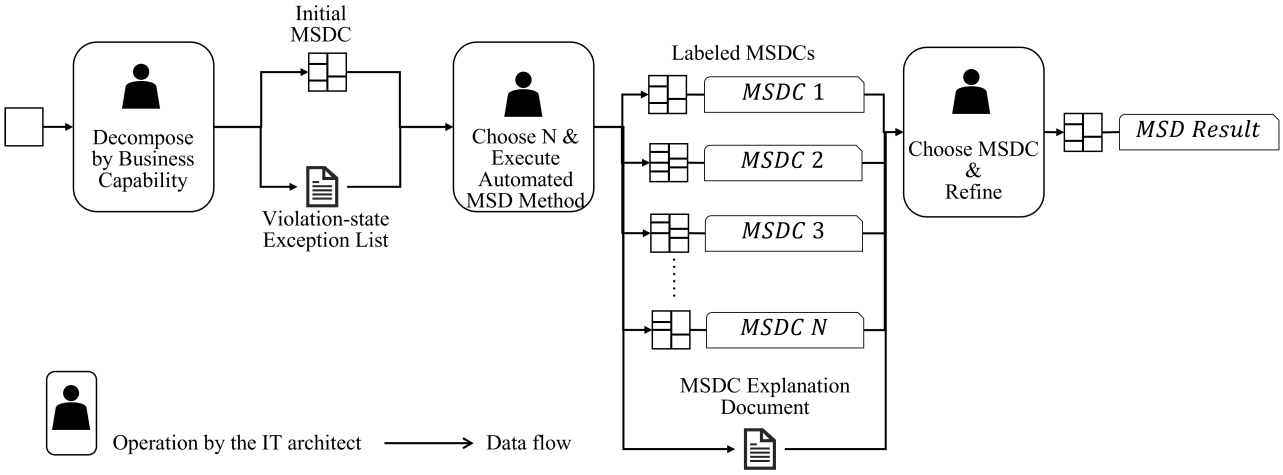


FIGURE 6. Flowchart of MSD using our method.

Algorithm 2 Generating Sets of Reference Line Vector

```
procedure RLVectors(d = 4, p)
    return GenRLVs(p, d, p)
end procedure

function GenRLVs(n, d, p)
    tmp_RLVs = {}
    if d==1 then
        ARRAY tmp_RLV[0]
        tmp_RLV[0] = n/p
        tmp_RLVs∪ = tmp_RLV
    else
        for i = 0, ..., n do
            suf_RLVs = GenRLVs(n − i, d − 1, p)
            for each suf_RLV ∈ suf_RLVs do
                ARRAY tmp_RLV[0..d − 1]
                tmp_RLV[0] = i/p
                for j = 1, ..., d − 1 do
                    tmp_RLV[j] = suf_RLV[j − 1]
                end for
                tmp_RLVs∪ = tmp_RLV
            end for
        end for
    end if
end function
```

the MyBatis Framework. The internet banking application is a Struts and Spring Framework application developed for a bank, which manages contractor account information and allows money transfer operations. This application uses a MySQL DB, and the program accesses the DB table using the iBatis Framework.

Table 3 illustrates the approach we used to extract the initial MSDC from each application. In the pet clinic application, API Gateway, class, and table were associated with the microservice on the basis of the method call from the @Controller class. For the internet banking and insurance contract management applications, each API Gateway, class, and table were associated with the microservice on the basis of the naming convention of each business function.

TABLE 5. Interpretation of Pearson correlation coefficient.

Absolute value of correlation coefficient	Interpretation
0.0–0.3	Weak linear relationship
0.3–0.7	Moderate linear relationship
0.7–1.0	Strong linear relationship

Table 4 presents the state of the initial MSDC. We considered the internet banking application acceptable for making method calls to Common microservice as Common microservice manages master data. In a similar vein, we considered the insurance contract management application acceptable for making method calls to Product, Contract, and Customer microservices as these are microservices that manage master data. We found 3, 28, and 13 violations in the pet clinic, internet banking, and insurance contract management applications, respectively.

B. EXPERIMENT

Data on the basis of evaluation items were collected in the applications. The number of iterations (Step A in Figure 2) was set to 100, and the Pareto-optimal MSDCs at each iteration were recorded. A PC with 32GB RAM and Intel®Core™i7-10610U CPU was used. The evaluation items are enumerated below.

EI1. Variance of the Pareto-optimal MSDCs at each iteration
EI2. Generation time of the Pareto-optimal MSDCs at each iteration
EI3. Generation rate of the selected MSDCs in Step B (Figure 2) at each iteration

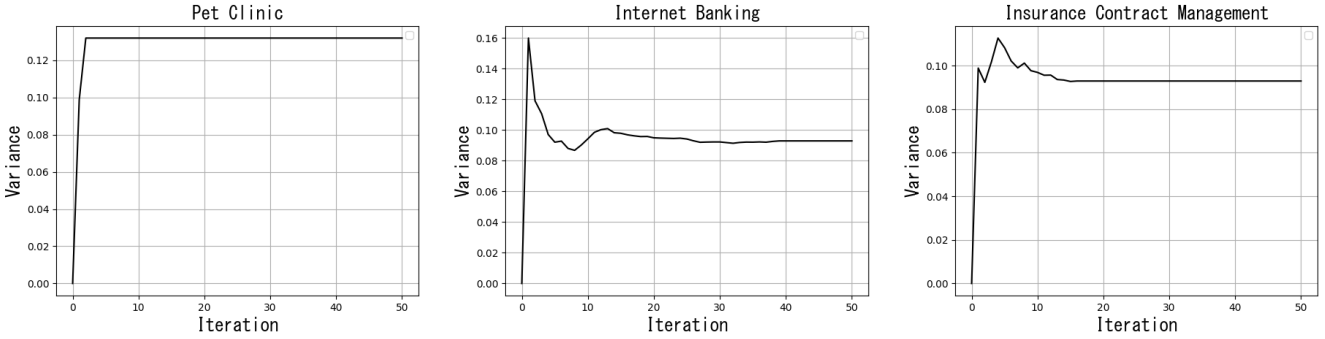


FIGURE 7. Variance of MSDCs across iterations.

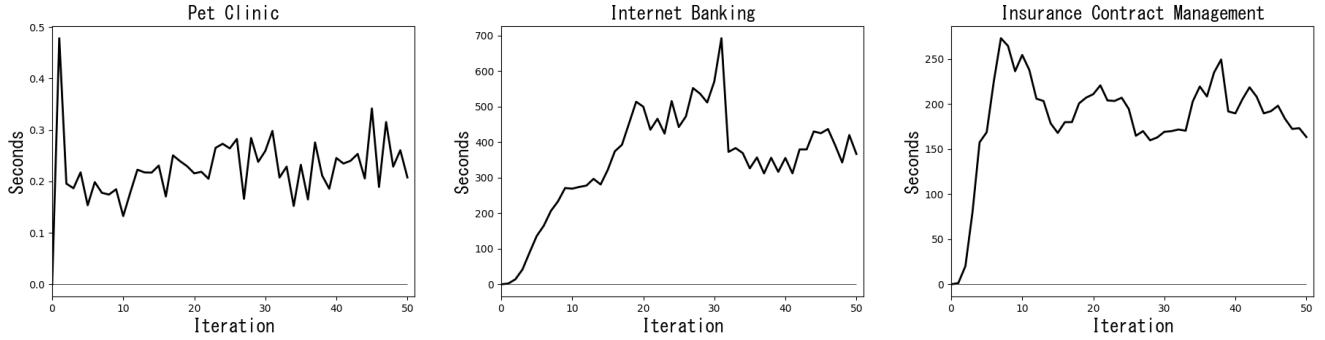


FIGURE 8. Time of generating MSDCs at each iterations.

EI4. Correlation between the reference-line vector and MSDC

EI5. Use of the proposed method by our IT architects

EI1–EI2 and EI3–EI5 were conducted to address research questions RQ1 and RQ2, respectively. EI3 and EI4 involved using four different patterns with 4, 10, 20, and 35 reference-line vectors, generated by setting p to 1, 2, 3, and 4 in Algorithm 2. The Pearson correlation coefficient [42] was used for EI4 to assess the linear association between two components. The coefficient is between 0 and 1, where a higher absolute value indicates a stronger correlation between the variables. Accepted guidelines for interpreting the correlation coefficient are provided in Table 5 as referenced from a previous study [43]. Thirty-five reference-line vectors were used with $p = 4$ in Algorithm 2 for EI5. The selection and fine-tuning of the MSD solution were conducted by experienced IT architects with over 10 years' experience in application design within the financial domain.

C. RESULTS

The results of EI1 are presented in Figure 7, where the horizontal axis represents the number of iterations and vertical axis shows the variance value. The graph displays the change in variance of the Pareto-optimal MSDCs as a black line. Across all three applications (pet clinic, internet banking, and insurance contract management), the variance increased steadily until a certain iteration then stabilized at a constant value. The number of MSDCs generated at the point of convergence was 4, 220, and 243 for the pet clinic, internet banking, and insurance contract management applications, respectively.

The results of EI2 are presented in Figure 8, where the horizontal axis represents the number of iterations and vertical axis shows the generation time in seconds. The graph shows the time evolution of the Pareto-optimal MSDCs as a black line. Across all three applications, the time increased steadily until a certain iteration. The generation time for the first iterations of the pet clinic, internet banking, and insurance contract management applications were 0.48, 2.20, and 1.32 s, respectively, while the maximum generation times at a specific iteration were 0.48, 692.50, and 273.18 s, respectively.

The results of EI3 are presented in Figure 9, illustrating the generation rate of the selected MSDCs across iterations. The graph showcases the impact of different reference-line vectors on the selection process, represented with the blue, green, red, yellow, and black lines. The blue line corresponds to MSDCs selected using 4 reference-line vectors, green line represents 10 reference-line vectors, red line represents 20 reference-line vectors, yellow line represents 35 reference-line vectors, and black line represents no reference-line vector usage. The generation rate refers to the level of concordance between the MSDCs selected by the reference line during the 100th iteration and those selected by the reference line in each iteration. For the pet clinic and insurance contract

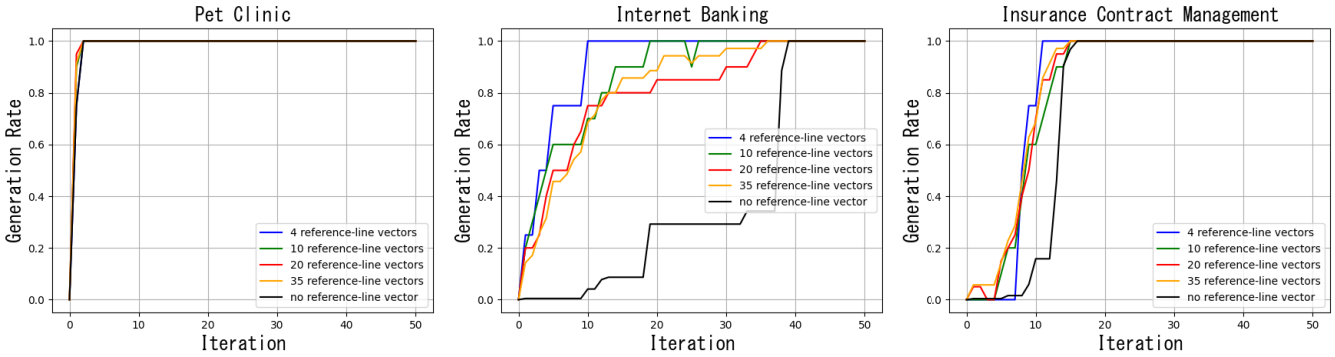


FIGURE 9. Generation rate of selected MSDCs in Step B (Figure 2) across iterations.

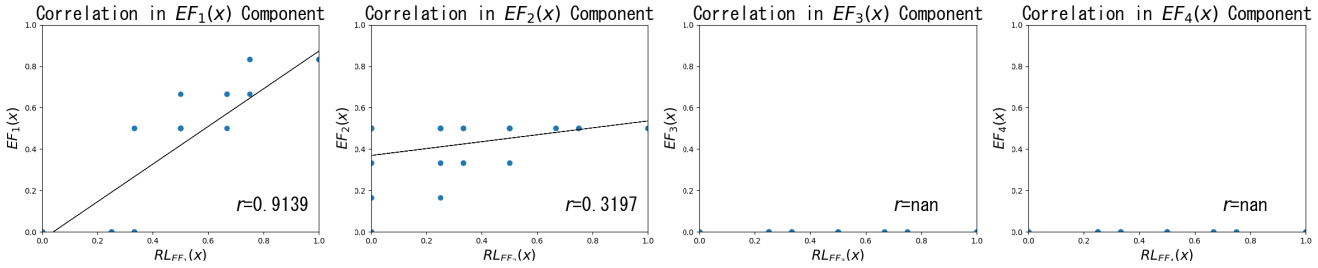


FIGURE 10. Correlation between reference-line vector and MSDC (pet clinic).

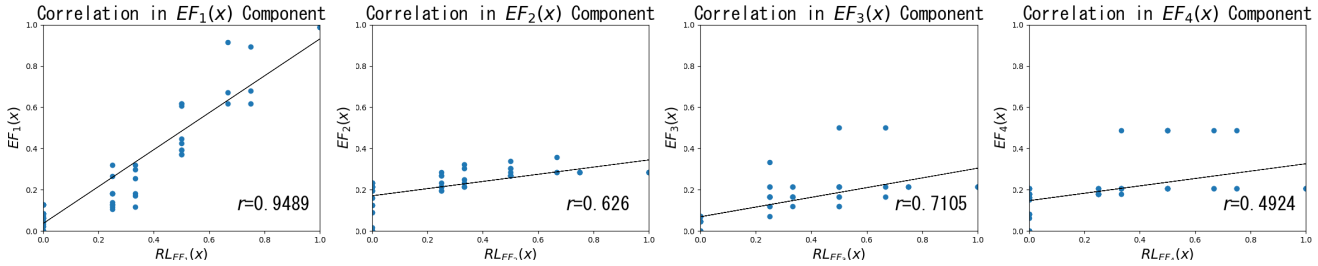


FIGURE 11. Correlation between reference-line vector and MSDC (internet banking).

management applications, there was no significant difference between the number of iterations required to achieve 100% generation for MSDCs selected by the reference-line vector and number of iterations needed for 100% generation without using reference-line vectors. However, for the internet banking application, the number of iterations required to attain 100% generation using reference-line vectors was lower than the number of iterations needed for 100% generation without using reference-line vectors.

The results of EI3 are presented in Figures 10, 11, and 12, where the horizontal axis represent the values of the $EF_1(x)$–$EF_4(x)$ components of the reference-line vector that selected the MSDCs, and the vertical axis show the values of the $EF_1(x)$–$EF_4(x)$ components of the MSDCs. To provide further clarification, we defined four reference-line functions (RL_{EF_1}–RL_{EF_4}) that takes an MSDC x as input and output the values of the $EF_1(x)$–$EF_4(x)$ components of the reference-line vector that selected the MSDC x, as shown in Table 6.

TABLE 6. Reference line functions.

Reference line function (x : MSDC)
$RL_{EF_1}(x)$. $EF_1(x)$ component value of reference-line vector that selected x.
$RL_{EF_2}(x)$. $EF_2(x)$ component value of reference-line vector that selected x.
$RL_{EF_3}(x)$. $EF_3(x)$ component value of reference-line vector that selected x.
$RL_{EF_4}(x)$. $EF_4(x)$ component value of reference-line vector that selected x.

Each graph plots the MSDCs selected by the reference-line vectors generated with p=1, 2, 3, and 4 in Algorithm 2. Figures 10, 11 and 12 illustrate the MSDCs for the pet clinic, internet banking, and insurance contract management applications, respectively.

The internet banking and the insurance contract management applications had correlation coefficients greater than 0.4 for all components, indicating a positive correlation

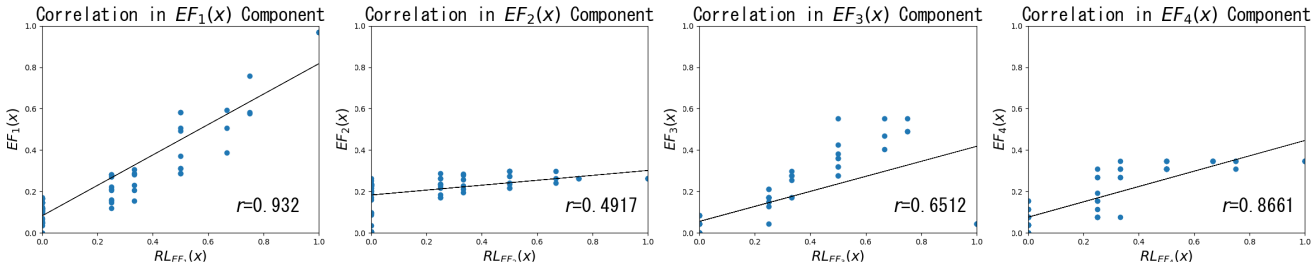


FIGURE 12. Correlation between reference-line vector and MSDC (insurance contract management).

TABLE 7. Chosen MSDC and final MSD result.

Subject application	Chosen MSDC	Reference-line vector	Final MSD result	Refinement done by IT architects
Pet Clinic	Pet & Owner, Welcome, Crash, Visit, Vet	(0.75, 0.25, 0, 0)	Customer, Visit, Vet	- Renamed Pet & Owner to Customer. - Deleted Welcome and Crash.
Internet Banking	Common, Bank, Login, Transfer, Contractor$_\beta$	(0, 0.25, 0.25, 0.50)	Balance, Fund Move, Bank, Customer, Login, Transfer, Contractor$_\alpha$	- Split Common into Balance and Fund Move. - Split Contractor$_\beta$ into Contractor$_\alpha$ and Customer.
Insurance Contract Management	New-Contract, Billing, Maintenance, Product, Contract, Customer	(0, 0.25, 0.25, 0.50)	New-Contract, Billing, Maintenance, Product, Contract, Customer	- No refactoring done.

between the component values of the MSDC and those of the reference-line vector. In the pet clinic application, all values of the correlation coefficients were 0.3 or higher, indicating at least a weak positive correlation between the component values of the MSDC and those of the reference-line vector. For the $EF_3(x)$ and $EF_4(x)$ components of the pet clinic application, correlation coefficients were not measured because the selected MSDC did not include V3 and V4.

The results of EI5 are presented in Table 7. The choice and fine-tuning of an MSDC were conducted by IT architects through a comparison between MSDCs using MSDC explanation document. The values of the reference-line vector listed in the MSDC explanation document were used to prioritize MSD policies.

In the pet clinic application, the selected MSDC took into account the higher importance of P2 compared with P1, as indicated with the reference-line vector (0.75, 0.25, 0, 0). Non-critical microservices, such as Welcome and Crash, were removed. Similarly, in the internet banking application, the selected MSDC reflected the highest priority given to P1, followed by P2 and P3, as indicated with the reference-line vector (0, 0.25, 0.25, 0.50). The Common microservice was subdivided into Balance and Fund Move microservices, while the Contractor$_\alpha$ microservice was divided into Contractor$_\beta$ and Customer microservices on the basis of a comparison with the adjacent MSDCs selected by the reference line (e.g. MSDC selected by the reference-line vector (0.25,0.25,0.25,0.25)). Finally, in the insurance contract management application, the selected MSDC reflected the highest priority given to P1, followed by P2 and P3, as indicated with the selected by the reference line reference-line vector (0, 0.25, 0.25, 0.50).

V. DISCUSSION

To evaluate the effectiveness of our MSD method, we addressed the three research questions mentioned earlier in the paper.

A. RQ1: WHAT IS THE IMPACT OF DEFINING AN MSD POLICY ON THE GENERATION OF PARETO-OPTIMAL MSDCS?

The generation of new MSDCs on the basis of the MSD policy decrease the number of MSDCs with lower semantic meaning, limiting the quantity of Pareto-optimal MSDCs. However, the generation time of new MSDCs are likely to increase depending on the number of program and data structures which violate MSD policies.

The results of EI1 indicate that the variance of Pareto-optimal MSDCs consistently increases up to a certain generation in all three applications. This suggests that by patternizing the modification methods and constraining their application scope, it becomes feasible to restrict the quantity of Pareto-optimal MSD solutions on the basis of MSD policies.

The results of EI2 indicate that the process of generating Pareto-optimal MSDCs during each iteration is contingent upon the number of violations in the initial MSDC. Possible factors contributing to the observed increase in generation time include the amount of methods, the number of DB tables, and the number of violations. However, when analyzing the time taken for the initial iteration in EI2, the ratio of generation time between the pet clinic and insurance contract management applications was less than the corresponding violation ratio between the same applications. Similarly, the generation time ratio between the pet clinic and banking applications was also less than the violation ratio associated

with these applications. These patterns suggest that the amount of methods and DB tables might minimally affect the lengthening of generation time, whereas violations likely play a more substantial role in this temporal expansion.

In summary, the generation of new MSDCs on the basis of violation detection and fix operation application limits the quantity of Pareto-optimal MSDCs. However, the generation time of new MSDCs are likely to increase depending on the number of violations in the initial MSDC.

B. RQ2: HOW DOES THE USE OF REFERENCE LINES IN THE GENERATION OF MSDCS CONTRIBUTE TO OUR PROPOSED METHOD AND WHAT ROLE DO THE COMPONENT VALUES OF THE REFERENCE-LINE VECTOR PLAY?

The use of reference lines accelerates the creation of MSD result by providing a framework for efficient choice and refinement of MSDC. The component values of the reference-line vector serve as indicators of the distinctive features of the MSDCs.

The results of EI3 indicate that when using the reference-line vector for choosing MSDCs, the generation process requires fewer iterations compared to generating Pareto-optimal MSDCs. This expedited generation is particularly beneficial in cases where the application exhibits numerous violations.

The results of EI4 highlight the correlation between the component values of the selected MSDCs and those of the reference-line vector. This correlation confirms that the component values of the reference-line vector effectively capture the distinctive features of the selected MSDC.

From the results of EI5, the prioritization of P1–P4 can be discerned by examining the values of the $EF_1(x)$ to $EF_4(x)$ components of the reference-line vector. This information provides insights into the relative importance of each policy and assists in determining the refinement for the chosen MSDC.

In summary, the use of reference lines enables IT architects to generate MSD solutions by providing a framework for choosing and refining MSDC using the reference-line vector. This approach enhances the decision-making process and enables the exploration of different trade-offs and considerations in the design of microservices.

C. RQ3: HOW DOES OUR METHOD PROVIDE SUPPORT FOR IT ARCHITECTS COMPARED WITH PREVIOUS METHODS?

Our method supports IT architects navigating the uncertainties associated with hyperparameters, particularly the determination of the number of partitions needed to transform the existing system into microservices. As outlined in Table 8, a comparison of hyperparameters between our method and previous methods, as enumerated in a previous study [25], highlights notable distinctions. Traditional methods often necessitate a predefined number of clusters or partitions for generating MSDCs. Yet, it is not common for IT architects to grapple with the challenge of pinpointing the optimal number of microservices tailored to a project's specific needs. Our method shifts the focus to mitigate this challenge: rather than pre-defining a fixed number of partitions, we introduce the concept of an initial MSDC as a hyperparameter. Based on the initial MSDC, a set of MSDCs, each of which may have a different number of partitions, are generated. This approach supports IT architects having knowledge about the business functions of the monolithic system, but having challenge determining the optimal number of partitions on the basis of both business functions and program structure.

TABLE 8. Comparison of hyperparameters with previous MSD methods.

Algorithm	Hyperparameter
mono2micro [21]	Number of clusters
FoSCI [30]	Number of clusters Number of iterations to run NSGA-II Population size for NSGA-II Parent size to use in NSGA-II Threshold for clustering to stop
MEM [23]	Maximum partition size Number of partitions
Bunch [28]	Number of partitions Initial population size for hill climbing Number of neighbors to consider in hill climbing iterations
CoGCN [44]	Number of clusters Loss function coefficients Number of hidden units in each layer Dropout rate
Our approach	Initial MSDC Number of MSDCs to select

VI. THREADS TO VALIDITY

Our method eliminates non-Pareto-optimal MSDCs at each iteration, which may lead to locally optimal solutions. However, even when using this method, the final MSD result will be further refined by IT architects. These architects may consider modification methods by comparing the selected MSDC with other MSDCs, rather than solely relying on the selected MSDC. By selecting MSDCs with diverse characteristics using reference-line vectors, even for locally optimal solutions, the objective of supporting MSD for IT architects can be achieved.

Our method generates MSDCs on the basis of four initial MSD policies, which stem from the analysis of program and data dependencies. Yet, there is room for new MSD policies that are not analyzable through such dependencies or initial MSDCs. For instance, Gysel et al. [18] defined 16 MSD policies, whereas our method encompasses only 4. In such scenarios, IT architects are required to exercise their judgment to manually select the MSDC that resonates best with their perspective and adjust it accordingly.

VII. CONCLUSION

We introduced a multi-objective optimization method using reference lines to guide the selection of MSDCs. By defining MSD policies, detecting violations, and implementing fix

operations, we effectively reduced the number of MSDCs. Through experimentation across three diverse applications—pet clinic, insurance contract management, and internet banking, we validated the efficacy of our proposed method in both generating MSDCs and assisting IT architects in MSD.

Our experimental findings provided key insights. First, the introduction of new MSDCs on the basis of defined policies reduced the increase of solutions with semantic relevance, limiting the number of Pareto-optimal MSDCs. However, the generation time of these MSDCs could be affected by the number of program and data structure violations against the MSD policies. Second, the implementation of reference lines notably accelerated the creation of the MSD result. The metrics of the reference-line vector effectively highlighted unique attributes of the selected MSDCs, streamlining choice and refinement of MSDC. This method proved instrumental for IT architects, presenting multiple MSDCs with distinct characteristics and enabling a comparison of their advantages and drawbacks.

In a comprehensive manner, our method addresses a significant challenge faced by IT architects: determining the optimal number of microservice partition for an existing monolithic system. By excluding the number of partition from the hyperparameters, our method offers a practical solution for IT architects facing difficulty predermining the number of partition on the basis of both business function and program structure.

Looking ahead, a deeper investigation into the generation time of MSDCs is necessary to further refine and optimize our method, especially for expansive and complex software projects. Tackling the temporal challenges in MSD will be essential for broadening the applicability and relevance of our method for large-scale software development.

REFERENCES

[1] N. Sam, *Building Microservices*, 2nd ed. Sebastopol, CA, USA: O'Reilly, 2021.

[2] J. Soldani, D. A. Tamburri, and W.-J. van den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *J. Syst. Softw.*, vol. 146, pp. 215–232, Dec. 2018.

[3] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices anti-patterns: A taxonomy," in *Microservices: Science and Engineering*. New York, NY, USA: Springer, 2020, pp. 111–128.

[4] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. Sebastopol, CA, USA: O'Reilly, 2019.

[5] N. Ford and M. Richards, *Fundamentals of Software Architecture: An Engineering Approach*, 1st ed. Sebastopol, CA, USA: O'Reilly, 2020.

[6] M. Li, T. Chen, and X. Yao, "How to evaluate solutions in Pareto-based search-based software engineering: A critical review and methodological guidance," *IEEE Trans. Softw. Eng.*, vol. 48, no. 5, pp. 1771–1799, May 2022.

[7] W. Mkaouer, M. Kessentini, A. Shaout, P. Koligheu, S. Bechikh, K. Deb, and A. Ouni, "Many-objective software remodularization using NSGA-III," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 3, pp. 1–45, May 2015.

[8] I. Candela, G. Bavota, B. Russo, and R. Oliveto, "Using cohesion and coupling for software remodularization: Is it enough?" *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, pp. 1–28, Aug. 2016.

[9] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, "Automated microservice identification in legacy systems with functional and non-functional metrics," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Mar. 2020, pp. 135–145.

[10] J. Branke, K. Deb, H. Dierolf, and M. Osswald, "Finding knees in multi-objective optimization," in *Parallel Problem Solving From Nature—PPSN XVII*. New York, NY, USA: Springer, 2004, pp. 722–731.

[11] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach—Part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Aug. 2014.

[12] A. Megargel, V. Shankararaman, and D. Walker, "Migrating from monoliths to cloud-based microservices: A banking industry example," in *Software Engineering in the Era of Cloud Computing*. New York, NY, USA: Springer, 2020, pp. 85–108.

[13] C.-Y. Fan and S.-P. Ma, "Migrating monolithic mobile application to microservice architecture: An experiment report," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Jun. 2017, pp. 109–112.

[14] C. Richardson, *Microservices Patterns: With Examples in Java*, 1st ed. Shelter Island, NY, USA: Manning, 2018.

[15] E. Ntentos, U. Zdun, K. Plakidas, and S. Geiger, "Semi-automatic feedback for improving architecture conformance to microservice patterns and practices," in *Proc. IEEE 18th Int. Conf. Softw. Archit. (ICSA)*, Mar. 2021, pp. 36–46.

[16] F. H. Vera-Rivera, E. Puerto, H. Astudillo, and C. M. Gaona, "Microservices backlog—A genetic programming technique for identification and evaluation of microservices from user stories," *IEEE Access*, vol. 9, pp. 117178–117203, 2021.

[17] L. Baresi, M. Garriga, and A. De Renzis, "Microservices identification through interface analysis," in *Proc. Service-Oriented Cloud Comput.*, 2017, pp. 19–33.

[18] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," in *Proc. Service-Oriented Cloud Comput.*, 2016, pp. 185–200.

[19] M. Kamimura, K. Yano, T. Hatano, and A. Matsuo, "Extracting candidates of microservices from monolithic application code," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2018, pp. 571–580.

[20] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, "Remodularization analysis for microservice discovery using syntactic and semantic clustering," in *Proc. CAISE*, 2020, pp. 3–19.

[21] A. K. Kalia, J. Xiao, C. Lin, S. Sinha, J. Rofrano, M. Vukovic, and D. Banerjee, "Mono2Micro: An AI-based toolchain for evolving monolithic enterprise applications to a microservice architecture," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 1606–1610.

[22] W. Jin, T. Liu, Q. Zheng, D. Cui, and Y. Cai, "Functionality-oriented microservice extraction based on execution trace clustering," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2018, pp. 211–218.

[23] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 524–531.

[24] L. Chen, M. Guang, J. Wang, and C. Yan, "Dynamic and static feature-aware microservices decomposition via graph neural networks," in *Proc. KSEM*, 2023, pp. 150–163.

[25] R. Yedida, R. Krishna, A. Kalia, T. Menzies, J. Xiao, and M. Vukovic, "Lessons learned from hyper-parameter tuning for microservice candidate identification," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2021, pp. 1141–1145.

[26] A. M. Saeidi, J. Hage, R. Khadka, and S. Jansen, "A search-based approach to multi-view clustering of software systems," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reengineering (SANER)*, Mar. 2015, pp. 429–438.

[27] D. Doval, S. Mancoridis, and B. S. Mitchell, "Automatic clustering of software systems using a genetic algorithm," in *Proc. 9th Int. Workshop Softw. Technol. Eng. Pract.*, Sep. 1999, pp. 73–81.

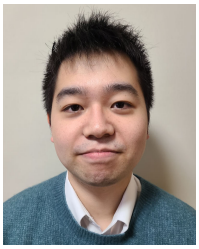
[28] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," *IEEE Trans. Softw. Eng.*, vol. 32, no. 3, pp. 193–208, Mar. 2006.

[29] K. Praditwong, M. Harman, and X. Yao, "Software module clustering as a multi-objective search problem," *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 264–282, Mar. 2011.

[30] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service candidate identification from monolithic systems based on execution traces," *IEEE Trans. Softw. Eng.*, vol. 47, no. 5, pp. 987–1007, May 2021.

[31] J. Ivers, C. Seifried, and I. Ozkaya, "Untangling the knot: Enabling architecture evolution with search-based refactoring," in *Proc. IEEE 19th Int. Conf. Softw. Archit. (ICSA)*, Mar. 2022, pp. 101–111.

[32] S. Khoshnevis, "A search-based identification of variable microservices for enterprise SaaS," *Frontiers Comput. Sci.*, vol. 17, no. 3, Jun. 2023, Art. no. 173208.
[33] C. Schröder, A. van der Feltz, A. Panichella, and M. Aniche, "Search-based software re-modularization: A case study at adyen," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, May 2021, pp. 81–90.
[34] L. Carvalho, A. Garcia, T. E. Colanzi, W. K. G. Assunção, M. J. Lima, B. Fonseca, M. Ribeiro, and C. Lucena, "Search-based many-criteria identification of microservices from legacy systems," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2020, pp. 305–306.
[35] L. Carvalho, A. Garcia, T. E. Colanzi, W. K. G. Assunção, J. A. Pereira, B. Fonseca, M. Ribeiro, M. J. de Lima, and C. Lucena, "On the performance and adoption of search-based microservice identification with toMicroservices," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2020, pp. 569–580.
[36] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, "Towards automated microservices extraction using muti-objective evolutionary search," in *Proc. 17th Int. Conf. Service-Oriented Comput.*, 2019, pp. 58–63.
[37] A. Trivedi, D. Srinivasan, K. Sanyal, and A. Ghosh, "A survey of multiobjective evolutionary algorithms based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 21, no. 3, pp. 440–462, Jun. 2017.
[38] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[39] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1989.
[40] I. Das and J. E. Dennis, "Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems," *SIAM J. Optim.*, vol. 8, no. 3, pp. 631–657, Aug. 1998.
[41] J. Bogner, S. Wagner, and A. Zimmermann, "Automatically measuring the maintainability of service- and microservice-based systems: A literature review," in *Proc. 27th Int. Workshop Softw. Meas. 12th Int. Conf. Softw. Process Product Meas.*, Oct. 2017, pp. 107–115.
[42] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. London, U.K.: Routledge, 1988.
[43] B. Ratner, "The correlation coefficient: Its values range between +1/−1, or do they?" *J. Targeting, Meas. Anal. Marketing*, vol. 17, pp. 139–142, May 2009.
[44] U. Desai, S. Bandyopadhyay, and S. Tamilselvam, "Graph neural network to dilute outliers for refactoring monolith application," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 72–80.



TAKAHIRO KINOSHITA received the M.E. degree from Tokyo Institute of Technology, Japan, in 2019. He is currently a Researcher with the Research and Development Group, Hitachi Ltd., Kanagawa, Japan. His current research interest includes software engineering for architecture and design.

HIDEYUKI KANUKA received the B.E. degree from Musashi Institute of Technology, Tokyo, Japan, in 2001, and the M.E. degree from Tokyo Institute of Technology, Japan, in 2003. Since 2003, he has been a Chief Researcher with the Research and Development Group, Hitachi Ltd., Kanagawa, Japan. In 2012, he was a Visiting Researcher with Stanford University, CA, USA. His current research interest includes software engineering for architecture and testing.