Research Article

A Cluster and Process Collaboration-Aware Method to Achieve Service Substitution in Cloud Service Processes

Qiang Hu and **Jiaji Shen**

School of Information Science and Technology, Qingdao University of Science and Technology, Qingdao 266061, China

Correspondence should be addressed to Qiang Hu; huqiang200280@163.com

Received 8 October 2019; Revised 25 June 2020; Accepted 10 July 2020; Published 1 August 2020

Academic Editor: Manuel E. Acacio Sanchez

Copyright © 2020 Qiang Hu and Jiaji Shen. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Some cloud services may be invalid since they are located in a dynamically changing network environment. Service substitution is necessary when a cloud service cannot be used. Existing work mainly concerned on service function and quality in service substitution. To select a more suitable substitutive service, process collaboration similarity needs to be considered. This paper proposes a cluster and process collaboration-aware method to achieve service substitution. To compute the process collaboration similarity, we use logic Petri nets to model service processes. All the service processes are transformed into path strings. Service vectors for cloud services are generated by Word2Vec from these path strings. Process collaboration similarity of two cloud services is obtained by computing the cosine value of their service vectors. Meanwhile, similar cloud services are classified as a service cluster. By calculating function similarity and quality matching, a candidate set for services substitution is generated. The service with the highest process collaboration similarity to invalid one in the candidate set is chosen as the substitutive one. Simulation experiments show the proposed method is less time-consuming than traditional methods in finding substitutive service. Meanwhile, the substitutive one has a high cooccurrence rate with neighboring services of the invalid cloud service. Thus, the proposed method is efficient and integrates process collaboration well in service substitution.

1. Introduction

With the promotion of cloud computing applications, a variety of cloud services with different functions are quickly registered in various cloud computing platforms [1]. Users can easily search and lease their expected cloud services in these cloud computing platforms. For example, “casi-cloud.com” is a cloud manufacturing platform providing manufacturing services. We can find that there are nearly 950,000 cloud services in the website and more than 8800T industrial data have been handled by these services in the end of August 2019 [2]. A new business application can be easily built by invoking these cloud services. To effectively address complicated service requests, we can assemble a group of cloud services as a composed service process with a specific business flow [3].

Service invocation is the most popular way to integrate existing cloud services in the network-based software

systems [4]. It can greatly reduce the time cost in constructing new business system. To integrate a cloud service, an appropriate service which can properly respond service request needs to be selected. Since there are a large number of cloud services in cloud platforms, service discovery is a time-consuming process. Current service discovery methods face a large searching space, and their seeking processes are tedious and inefficient [5–7].

In the complicated and precarious network environment, some cloud services may be invalid during their invoking processes [8]. A substitutive service need to be searched once any of the component services is unavailable in service-oriented business systems [9]. Many service replacement methods are inefficient. The main reason of low efficiency is service substitution faces a large searching space. The existing work mainly concerns on service function and quality in service substitution. These methods can find a service to replace the invalid one. The substitutive service is

equivalent in service function and quality. However, it may not be able to cooperate with other services as well as the invalid one. The main reason is that process collaboration is not considered in service substitution [10].

Aiming at finding a quick and more reasonable substitutive service, we propose a cluster and process collaboration-aware method to achieve service substitution. To improve the efficiency of service discovery and substitution, we cluster cloud services with the same or similar functions as a group, named as a service cluster. The clustering mechanism can reduce service searching space. It can improve the efficiency of service discovery and substitution. We also take process collaboration of the component services into consideration. The candidate service with the highest process collaboration similarity to invalid one is recommended to apply service substitution. The main contributions of this paper are as follows:

- (1) We introduce clustering mechanism to reduce service searching space. The efficiency of service discovery and substitution is prominently increased.
- (2) A method to evaluate process collaboration similarity is proposed. Service processes are transformed into path strings. We train service vectors for cloud service by Word2Vec based on these path strings. Then, process collaboration similarity is obtained by computing the cosine value of service vectors.
- (3) Service function, quality, and process collaboration are comprehensively considered to achieve service substitution. The proposed cluster and process collaboration-aware method is obviously superior to the existing methods in service substitution.

The rest of this paper is organized as follows: Section 2 introduces the related work about service substitution; the concept of service cluster and service response schema based on service clusters is presented in Section 3; how to substitute cloud service based on the cluster and process collaboration-aware method is proposed in Section 4; Section 5 presents simulation experiments; and Section 6 concludes this work.

2. Related Work

Finding an appropriate service for the invalid one is a key work in service substitution. Thus, the existing service discovery methods can offer an important reference for the research of service substitution. Cheng presents a diversified keyword search approach on service connection graphs. This method can satisfy the various possible requirements underlying a given keyword query [11]. Zhang defines a service composition context model based on three types of parameter correlations between service input and output parameters. The similarity between any two services is measured using the PersonalRank and SimRank++ algorithms by the composition context model [12].

Chen proposes a new measure of semantic similarity integrating multiple conceptual relationships for web

service discovery. The new measure enables more accurate service-request comparison by treating different conceptual relationships in ontologies such as is-a, has-a, and antonymy differently [13]. A comprehensive ontology has been developed to provide a standardized semantic specification of cloud services based on their functional features and nonfunctional features in [7]. The authors present an intelligent cloud service discovery framework based on these ontology concepts to identify cloud service. The average amount of error expected to identify a service by using the proposed framework is 11% compared to 31% by using the cloud service discovery solution. Hierarchical Dirichlet processes (HDP) model and personalized PageRank algorithm are used to achieve a two-stage model for cloud service recommendation by integrating the information of service descriptive texts and service tags [14]. Nabli proposes a self-adaptive semantic focused crawler based on latent Dirichlet allocation (LDA) for efficient cloud service discovery [15]. A method to learn features from service descriptions by using variational autoencoders is proposed by Lizarralde. It achieves significant gains compared to both word embeddings and classic latent features modelling techniques [16]. The above methods are the latest service discovery methods in recent three years. We can see that service context, comprehensive ontology about cloud services, and vector-based service similarity calculation are more concerned in service discovery.

In the domain of service substitution, researchers have also presented some effective methods. For example, Gong employs a cloud model to compute the QoS uncertainty to determine dynamic substitute targets. By targeting substitutions, the reconfigured web service will better satisfy users' requirements [17]. Three rules are provided to establish the compatibility and substitution of service operation interfaces [18]. The experiments to show service substitute identification based on the proposed framework achieve a best precision of 85%. By recording execution context data and mining the execution context conditions, an execution context-aware approach for web service substitution is proposed in [19].

Santhanam uses preference networks to represent and reason about preferences over nonfunctional properties in service substitution [20]. The proposed method is independent of the specific formalism used to represent functional requirements of a composite service as well as the specific algorithm used to assemble the composite service. By computing similarity degree between interface data and analyzing critical paths, Gao presents a method to check the data consistency for the dynamic replacement of service process [21]. This method provides fundamental theory guidance to enhance the credibility of the service process in the modern service industry. In recent research work, Sara et al. presented a similarity network for web services operations substitution [22]. The nodes represent the operations of the web services. A link joins two similar operations according to some relationships defined between them. The constituted network responds to the substitution best and much easier than existing works.

The aforementioned methods must go through a large number of cloud services to find the substituted one in service substitution. Most of previously mentioned methods are time-consuming. In [23], Wu proposes deploying a web service cluster to perform service substitution. Service cluster contains a logic service and a set of concrete services, and these concrete services have functional equivalence or compatibility. Du converts a service cluster into a service cluster net unit. And it is used to analyze whether the services in the cluster can satisfy some service requests [24]. However, service clusters in their methods are restricted with the same interfaces. They can reduce the searching space, but the flexibility of substitution still needs improvement.

The existing work mainly concerned on service function and quality in service substitution. Response time and substitution recall rate are the main evaluation indexes. We can also find some researchers give theoretical analysis to prove the feasibility and effectiveness of their approach. Few studies have taken process collaboration in consideration. The substitutive service will show better cooperation effect with other services once we add process collaboration relations in service substitution. In this paper, we introduce process collaboration similarity into service substitution and investigate service cooccurrence rate to show the benefits of our method.

3. Similarity Computation of Process Collaboration

A service process is composed of several cloud services. These cloud services cooperate to accomplish service request from tenants. We can obtain the collaboration similarity from the existing service processes.

There are two factors to determinate collaboration similarity for cloud services: one is cooccurrence rating, and another is the process distance. Two cloud services will have a higher collaboration similarity if they simultaneously appear more times than other services. Moreover, two services will also have a higher collaboration similarity once they are with a smaller process distance.

3.1. Path Strings of Service Processes. In this study, we first convert service processes into path strings. Then, we train service vectors for all the cloud services in these path strings by Word2Vec. Finally, we compute the collaboration similarity based on these service vectors. To obtain path strings, we use service nets to formally model cloud service processes. In service nets, logic Petri net is employed to describe the business flow. Now, we give the definition of logic Petri net.

Definition 1 (logic Petri net [25])

$LPN = (P, T, F, I, O, M)$ is a logic Petri net, where

- (1) P is a set of places
- (2) T includes three subsets of transitions, i.e., $T = T_D \cup T_I \cup T_O$, where T_D denotes a set of traditional

transitions, T_I denotes a set of logic input transitions, and T_O denotes a set of logic output transitions

- (3) F is a flow relation, i.e., a set of directed arcs $F \subseteq (P \times T) \cup (T \times P)$
- (4) I and O are mapping functions between logic input transitions and logic input expressions, i.e., $\forall t \in T_I, I(t) = f_I(t); \forall t \in T_O, O(t) = f_O(t)$
- (5) $M: P \rightarrow \{0, 1\}$ is a marking function, $\forall p \in P$, and $M(p)$ denotes the token count in p
- (6) Transition firing rules
 - (a) $\forall t \in T_D$, and the firing rules of t are the same as in PNs
 - (b) $\forall t \in T_I$, and t is enabled only if $f_I(t)|_M = \bullet T_\bullet$, where $\bullet T_\bullet$ denotes the logic value “true”. $M[t > M']$, where $\forall p \in \bullet t, M'(p) = 0; \forall p \in t^\bullet, M(p) = 0$, and $M'(p) = 1$; and $\forall p \notin t \cup t^\bullet, M'(p) = M(p)$
 - (c) $\forall t \in T_O$, and t is enabled only if $\forall p \in \bullet t: M(p) = 1$. $M[t > M']$, where $\forall p \in \bullet t: M'(p) = M(p) - 1; \forall p \notin t \cup t^\bullet: M'(p) = M(p)$; and $\forall p \in t^\bullet$ must satisfy $f_O(t)|_{M'} = \bullet T_\bullet$; i.e., $t^\bullet$ must satisfy the logic output expression $f_O(t)$ at M'

Definition 2 (service net)

A service net $SN = (LPN, i, o, L)$ is a labelled logic Petri net, where

- (1) LPN is the process model of a service process, where T_D denotes the component services. $P = P_c \cup P_d$, P_d is a set of data places interacting with the external services, and P_c is a set of control places representing the states of the service process
- (2) i is the initial place, and o is the terminal place of a service process, with $\bullet i = o^\bullet = \emptyset$
- (3) $L: T \rightarrow \Theta$ is a mapping function, where Θ is a set of cloud service names

Definition 3 (preset/postset)

For a service net $LPN, x \in P \cup T, \bullet x = \{y | (y, x) \in F\}$ is called a preset of x , and $x^\bullet = \{y | (x, y) \in F\}$ is called a postset of x .

To get the preset and postset of x , we introduce two operations, π and τ , to compute $\bullet x$ and $x^\bullet$. In this study, $\pi(x) = \bullet x$ and $\tau(x) = x^\bullet$.

Definition 4 (paradigm of logic expressions)

$F = \{f_1, f_2, \dots, f_n\}$ is a group of service nets, and t_o and t_i are the logic transitions to link F , i.e., $\forall f_j: \pi(f_j, i) = \{t_o\} \wedge \tau(f_j, o) = \{t_i\}$. The paradigm of logic expressions is defined as follows:

- (1) $O_\vee: O(t_o) = f_{1,i} \wedge \neg f_{2,i} \wedge \dots \wedge \neg f_{n,i} \vee f_{1,i} \wedge f_{2,i} \wedge \dots \wedge f_{n,i} \vee \neg f_{1,i} \wedge \neg f_{2,i} \wedge \dots \wedge \neg f_{n,i}$
- (2) $O_\wedge: O(t_o) = f_{1,i} \wedge f_{2,i} \wedge \dots \wedge f_{n,i}$
- (3) $I_\vee: I(t_i) = f_{1,o} \vee f_{2,o} \vee \dots \vee f_{n,o}$
- (4) $I_\wedge: I(t_i) = f_{1,o} \wedge f_{2,o} \wedge \dots \wedge f_{n,o}$

A service net for online shopping is provided in Figure 1. The service process described by this service net is initialized by inquiring some merchandises. If the query fails, a service is presented to show failure information. If the online seller can provide these merchandises, another service process which can purchase merchandises will be triggered. As we all know, either payment before receipt or receipt before payment can be both supported in the online trade. So, two subprocesses are concurrently presented. One is “reserve-defray-delivery,” and the other is “reserve-delivery-defray.” However, the logic expression labelled on transition t_o' is $p_5 \wedge \neg p_6 \vee \neg p_5 \wedge p_6$, and it means that only one of the places in p_5 and p_6 can be assigned one token. Thus, only one service process can be performed. Similarly, t_i' is labelled with a logic expression $p_{11} \vee p_{12}$, and it means that p_{13} can get a token once p_{11} or p_{12} has obtained the token. How to construct service nets for service processes can be found in our previous work [26].

To compute service vectors for the component services in the service processes, we convert the paths of service nets into strings, named path strings. A lot of path strings can be acquired from the existing service nets. The symbols in these path strings can form a corpus. Then, the tool Word2Vec is employed to train service vectors for the component services by utilizing these path strings mapping from the service nets. Finally, we can obtain collaboration similarity for two cloud services by computing the cosine similarity based on their service vectors. Relevant introduction about Word2Vec can be referred from [27]. The following section presents how to generate path strings and give an algorithm to compute service vector for each cloud service.

There are four types of basic process structures in the service nets: sequence, choice, parallel, and loop. The concept “one-fold service process” is proposed in our previous work [25]. In the one-fold service process, logic expressions labelled on logic transitions in the structures of choice and parallel must strictly follow Definition 4. Meanwhile, nesting structures must be not found in the one-fold structure. Four types of basic one-fold structures are illustrated in Figure 2. A merge-reduced method is introduced to generate a path string in this study. In the merging phase, all basic one-fold structures are mapped into a path string. The one-fold structures (a), (b), (c), and (d) can be merged as the path strings “ $t_1 t_2 t_3 \dots t_n$,” “ $O_{\wedge} t_1 \otimes t_2 \otimes t_3 \dots \otimes t_n \otimes I_{\wedge}$,” “ $O_{\vee} t_1 || t_2 || t_3 || \dots || t_n || I_{\vee}$,” and “ t_i ,” respectively.

There are very few one-fold structures in the service nets in practice. If a transition t_i is replaced by a service process sp_i , we should first merge all the transitions in sp_i and then replace t_i by the path string generated from sp_i . In the reducing phase, we link all the path strings obtained from the four types of process structures to generate a path string for a service net. The path string of the service net in Figure 1 is generated as “ $t_1 t_o t_2 || t_o' t_3 t_4 t_5 || t_3 t_5 t_4 || t_i' || t_i$ ” by this method. Here, service names have been mapped into symbols as $\{t_1$: query, t_2 : query fail, t_3 : reserve, t_4 : defray, t_5 : delivery $\}$. Details about how to obtain path string for a service net are presented in Algorithm 1.

FIGURE 1: Service net for online shopping.

3.2. Computation of Process Collaboration Similarity.

Given a group of cloud services, process collaboration similarity is used to evaluate what extent two cloud services can cooperate with other ones. Normally, two cloud services with high-process collaboration similarity means that they may have more partner cloud services in service processes. Since service vectors for cloud services can be trained from the path strings, the process collaboration similarity of two cloud services can be obtained by computing the cosine similarity of their service vectors.

Assume there are two resource pools: the cloud service clusters pool (CSCP) and service net pool (SNP). All the service clusters and cloud services are organized in CSCP. Meanwhile, the existing cloud service processes have been transformed into service nets and stored in SNP. Algorithm 2 provides a method to generate word vectors and service vectors for cloud services and service clusters in CSCP.

In Algorithm 2, we first construct two corpus CP_1 and CP_2 . CP_1 consisted of the description sentences of all the cloud services. All the path strings of service nets are gathered in CP_2 (lines 1 to 10). For the cloud service and service cluster, CP_1 is used to train word vector for words in function description item $D.O_p$ and $D.T_h$. These word vectors are used to compute the function similarity of cloud

FIGURE 2: Basic one-fold structures of service nets: (a) sequence; (b) parallel; (c) choice; (d) loop.

```

Input: service net SN;
Output: path string of SN;
(1)  $t_1 = t(SN.i)$ ;
(2)  $PS = t1$ ;
(3)  $t_c = t1$ ;
(4)  $t_n = t(\tau(t_1))$ ;
(5) while ( $t_n \neq \text{Null}$ )
(6) { if ( $t_c \in SN.T_s \wedge t_n \in SN.T_o$ )  $PS = PS + t_n$ ;
(7) if ( $t_c \in SN.T_s \wedge t_n \in SN.T_o$ )
(8) {  $m = |\tau(t_n)|$ ;
(9) For  $j = 1$  to  $m$ 
(10) obtain a place  $p$  in  $\tau(t_n)$  and build service net  $\text{SubSN}_j$  with  $\text{SubSN}_j.i = p$ ;
(11)  $sp_j = \text{PathString\_Generate}(\text{SubSN}_j)$ ;
(12) if ( $I(t_n) = O_v$ )  $PS = PS + sp_j + ||$ ;
(13) if ( $I(t_n) = O_\wedge$ )  $PS = PS + sp_j + \emptyset$ ;
(14) End for
(15) if ( $t_c \in SN.T_s \wedge t_n \in SN.T_o \wedge j = m$ )  $PS = PS + t_n$ ;
(16)  $t_c = t_n$ ;  $t_n = t(\tau(t_c))$ ; }
(17) return (PS);
(18) }

```

ALGORITHM 1: PathString_Generate (SN).

service and service cluster in finding candidate service set (see lines 11 to 13). In line 14, CP_2 is used to train service vector for cloud services. Since CP_2 consists of the path strings, the service vector trained by CP_2 can be adopted to calculate the collaboration similarity.

Definition 5 (process collaboration similarity)

Assume S is a set of cloud services. Let PS be the set of path strings of all the services in S . For two service S_i and S_j in S , P_i and P_j are service vectors of S_i and S_j which are trained by the corpus PS . The collaboration similarity of S_i and S_j is defined as $\text{CollSim}(S_i, S_j)$. $\text{CollSim}(S_i, S_j) = (P_i \cdot P_j) / (\|P_i\| \times \|P_j\|)$.

Notice that we omit the semantic of symbols in path strings, and only the positional adjacency of different

symbols is considered to train the vectors. Thus, we use the serial numbers of cloud services to generate path strings in practice.

4. Service Substitution Based on Clustering and Process Collaboration-Aware Method

In this section, we first introduce the concept of service cluster, present the service response schema based on service clusters, and then propose the cluster and process collaboration-aware method to achieve service substitution.

4.1. Service Response Schema Based on Service Clusters. Some similar definitions to describe a group of web services are put forward in the existing research, such as service pool

Input: services and service clusters in CSCP;
Output: vectors for the services and service clusters

- (1) Construct two corpus CP1 and CP2;
- (2) $CP1 = CP2 = \Phi$;
- (3) For each cloud service S in CSCP
- (4) Obtain the sentence send in S.D.Ft and $CP1 = CP1 \cup \{\text{Send}\}$;
- (5) End for
- (6) For each service net SN in SNP
- (7) $ps = \text{PathString_Generate}(SN)$;
- (8) Delete the symbol || and $\otimes$ from ps
- (9) $CP2 = CP2 \cup \{ps\}$;
- (10) End for
- (11) For each cloud service S and service cluster SC in CSCP
- (12) Train word vector S.WOp and S.WTh for the word in S.D.Op and S.D.Th by CP1;
- (13) Train word vector SC.WOp and SC.WTh for the word in SC.D.Op and SC.D.Th by CP1;
- (14) Train service vector PS for S by CP2;
- (15) End for
- (16) Return (S.WOp, S.WTh, SC.WOp, SC.WTh and P_s);

ALGORITHM 2: ServiceVector_Generate (CSCP).

[28], service class [29], and service cluster [26, 30]. Cloud services in above concepts are required with the same input and output parameters. Thus, they have little flexibility in service substitution because they can only achieve service migration with same interfaces.

In this paper, we do not require all the cloud services in a service cluster with the same interfaces. The definitions of cloud service and service cluster are formally defined as follows.

Definition 6 (cloud service)

A cloud service is a 6-tuple $C_{ls} = (N, D, I, O, Q, L)$, where

- (1) N is the serial number of cloud service in cloud service platform
- (2) D is a function description of the cloud service
- (3) I and O are the sets of input and output parameters, respectively
- (4) Q is a set of quality parameters
- (5) L is the URI of the cloud service

Function description of a cloud service is defined as $D = \langle O_p, T_h, F_t \rangle$. Here, O_p , T_h , and F_t are the operation, theme, and function text of a cloud service, respectively. For example, a weather forecast service is set as $D = \langle \text{query, weather, "the service can provide the weather forecast, users present the city and date, and then, the service can return temperature, humidity, ultraviolet intensity, and wind speed."} \rangle$.

As we known, service quality is an important factor to evaluate a cloud service. There are many common attributes in cloud services, such as response time, cost, and reliability. Besides, there may be some other quality attributes related to the practical application domain of cloud services. For example, manufacturing cycle and the level of after-sales service are more concerned by the tenants in cloud manufacturing.

Here, all these attributes are defined as quality parameters. We formally define it as $Q = \{q_i\}$, $q_i = (n, c, v, u)$, where n is the name of quality parameter, c is a comparison operator, v is the value of the parameter, and u is the unit of quality parameter. If a cloud manufacturing service is assigned as $q = (\text{manufacturing cycle, } \leq 5 \text{ day})$, it means that the manufacturing cycle is no longer than five days.

Definition 7 (service cluster)

A service cluster is a 6-tuple $\text{Sec} = (N, D, I, O, S, Q_g)$, where

- (1) N is the serial number of cloud service in cloud service platform
- (2) D is a function description of a service cluster
- (3) I and O are the sets of input and output parameters
- (4) $S = \{\text{cls1, cls2, } \dots, \text{cls}_n\}$ is the set of component services in a service cluster, where cls_i is a cloud service with $1 \leq i \leq n$
- (5) $Q_g = \{q_i\}$, where $q_i = \{n, c, [v_{\min}, v_{\max}], u\}$, and $v_{\min}$ and $v_{\max}$ represent the upper and lower bound values of q_i , respectively

Figure 3 shows the architecture of service response schema based on service clusters. Cloud services published by service providers are stored in the physical resource layer. Service cluster is a mapping collection of these services, and all the service clusters constitute the virtual resource layer [23].

The tenant request is modelled and submitted in the business model layer. It can be responded as two ways: single service or service composition. To respond to tenant request, we can find another service to substitute the invalid one in its responding service cluster. In majority of cases, the searching space is the volume of the corresponding service cluster; thus, the efficiency of service substitution can be greatly improved.

FIGURE 3: The architecture of service response schema based on service clusters.

4.2. Service Substitution. A cluster and process collaboration-aware method is proposed to achieve service substitution in this paper. The method can be divided into two steps: (1) we find a candidate service set for substitution based on service clusters. All these candidate services can replace the invalid one in view of service function and quality. (2) We compute the vector similarity between candidate services and invalid one so as to obtain collaboration intensity. By comprehensive consideration of function, quality, and collaboration, a cloud service with the highest similarity for the invalid one is selected to perform service substitution.

Definition 8 (functional similarity)

S_1 and S_2 are two cloud services. Let W_{Op1} and W_{Th1} be word vectors of $S_1.D.O_p$ and $S_1.D.T_h$, respectively, where $i = 1, 2$. The functional similarity of S_1 and S_2 is defined as $\text{FuncSim}(S_1, S_2)$. $\text{FuncSim}(S_1, S_2) = (1/2) * ((W_{Op1} \cdot W_{Op2} / \|W_{Op1}\| \times \|W_{Op2}\|) + (W_{Th1} \cdot W_{Th2} / \|W_{Th1}\| \times \|W_{Th2}\|))$.

Two cloud service S_1 and S_2 are called functional equivalence if $\text{FuncSim}(S_1, S_2) \geq \delta$. Here, δ is a threshold value. Meanwhile, the functional equivalence for two cloud services is denoted by $S_1 \leftrightarrow S_2$.

Definition 9 (parameter compatibility)

P_x and P_y are two parameters. Parameter compatibility of P_x and P_y is the replaceable degree of P_x and P_y , denoted as $\text{PC}(P_x, P_y)$.

Parameter compatibility is used to evaluate whether two groups of parameters can replace each other. It is divided into three levels in this study. To differentiate each level, we introduce three functions $\text{Num}(P)$, $\text{type}(P)$, and $\text{value}(P)$ to represent the amount, type, and value of parameter P , respectively. The partition rules of parameter compatibility are formally described as follows:

- (1) $\text{PC}(P_x, P_y) = L1$ if $\text{Num}(P_x) \leq \text{Num}(P_y)$ and $\forall m_i \in P_x \exists n_j \in P_y: m_i \leftrightarrow n_j \wedge \text{Type}(m_i) = \text{Type}(n_j)$
- (2) $\text{PC}(P_x, P_y) = L2$ if $\text{PC}(P_x, P_y) = L1 \wedge \text{PC}(P_y, P_x) = L1$
- (3) $\text{PC}(P_x, P_y) = L3$ if $\text{PC}(P_x, P_y) = L1$ and $\forall m_i \in P_x, \exists n_j \in P_y: \text{value}(m_i) \subseteq \text{value}(n_j)$

$\text{PC}(P_x, P_y) = L1$ means that P_x is a subset parameter of P_y , and it is symbolically represented as $P_x \propto P_y$. Similarly, $\text{PC}(P_x, P_y) = L2$ means that P_x is a isomorphic parameter of P_y and it is symbolically represented as $P_x \leftrightarrow P_y$. Meanwhile, $\text{PC}(P_x, P_y) = L3$ is denoted as $P_x \geq P_y$.

Definition 10 (quality score)

$S = \{\text{cls}_1, \text{cls}_2, \dots, \text{cls}_m\}$ is the component cloud services in a service cluster. Assume that each service in S has n quality parameters, i.e., $\langle q_{i1}, q_{i2}, \dots, q_{in} \rangle$ is the quality parameters of cls_i . The quality score of cls_i is defined as $\text{Qscore}(\text{cls}_i)$:

$$\text{Qscore}(\text{cls}_i) = \sum_{j=1}^n q_{ij} * w_j, \sum_{j=1}^n w_j = 1, \quad (1)$$

$$q_{ij} = \begin{cases} \frac{q_j^{\max} - q_{ij}}{q_j^{\max} - q_j^{\min}}, & \text{if } q_j^{\max} - q_j^{\min} \neq 0, \\ 1, & \text{if } q_j^{\max} - q_j^{\min} = 0, \end{cases} \quad (2)$$

$$q_{ij} = \begin{cases} \frac{q_{ij} - q_j^{\min}}{q_j^{\max} - q_j^{\min}}, & \text{if } q_j^{\max} - q_j^{\min} \neq 0, \\ 1, & \text{if } q_j^{\max} - q_j^{\min} = 0. \end{cases} \quad (3)$$

Quality parameters of cloud service can be divided into two types: positive parameters and negative parameters. Positive parameters will be attached with a higher quality when they are assigned a bigger value. On the contrary, negative parameters are attached with a

lower quality when they are assigned a bigger value. Formula (2) is adopted to scale positive parameters, while formula (3) is utilized to scale negative parameters. Quality score can be computed by formula (1) after all quality parameters have been normalized.

In Algorithm 3, CSCP is a cloud service cluster pool. All the cloud services and service clusters are stored in CSCP. In line 1, we initialize two empty sets, i.e., CS_R and cs_r . All the possible cloud service clusters which can provide the similar function are enrolled into CS_R . The candidate service set for substitution is represented as cs_r . By traversing CSCP, we can obtain every cloud service cluster in line 2. Functional similarity between each service cluster and the invalid service Se is computed, and the service cluster will be added to CS_R if the function similarity is larger than a threshold δ .

For each component cloud service in $CS_R.S$, we apply interfaces and quality matching in line 6 and line 7. In the level of interface, we know that cs_1 can replace cs_2 if the input parameters of cs_1 are the subset parameters of cs_2 's input parameters, while the output parameters of cs_2 are the subset parameters of cs_1 's output parameters. Meanwhile, the quality parameters of cs_1 should also provide a wide range value than cs_2 . For $cs \in CS_R.S$ and Se , it can be formally described as $cs.I \propto Se.I \wedge Se.O \propto cs.O \wedge cs.Q \geq Se.Q$. Finally, the candidate service set for substitution of Se is obtained in line 8 as the set cs_r .

From line 10 to line 12, we give a comprehensive scoring method to rank the service quality and collaboration similarity. Here, the weights α and β can be set according to the tenants. Both α and β are assigned as 0.5 in this paper. The top rating cloud service will be returned to substitute the invalid service Se in lines 13 and 14.

Compared with traditional service discovery or substitution, our method needs to add service clusters. The number of service clusters will directly affect resource consumption. To verify how the granularity of service cluster affects service lookup time, we have grouped the 5000 cloud services into 50, 100, 200, 400, 600, 800, and 1000 service clusters, respectively. We find that when the number of service clusters is about 20%–40% of the total number of services, the service discovery is with a high efficiency.

In previous work, we have discussed the impact of service cluster granularity on service discovery from three aspects: quantity, structure, and quality [31]. However, we cannot give a specific granularity value on which the service discovery is in a highest efficiency. It is because we are unable to determine the size of the number of services in each service cluster. Normally, we can conclude from experiments that the scenario where the number of service clusters is about 20%–40% of the total number of services is the best granularity. Thus, we think the resource consumption will increase by 20%–40% by introducing service clusters in our method. In addition to these, we need to add a 200-dimensional vector for each service and its functional description to calculate the functional similarity. Of course, similar resource consumption also exists in other vector-based service similarity calculation work [14–16].

5. Simulation Experiments

Simulation experiments are conducted to show the efficiency of the proposed method. Hardware for the computer is as follows: CPU is i5-8500 with 3.0 GHz, six cores. Memory is 16 G. Graphics card is GTX1060 with 6 G. Simulation program is designed by Java.

“Casicloud.com” is a famous industrial Internet platform of China. A large number of cloud manufacturing services were registered in this platform. We crawl 3780 cloud services from “casicloud.com.” These cloud services are about the same manufacturing domain. Four hundred cloud services are randomly selected, and we manually build two to five similar services for each selected cloud service. The total number of cloud services in simulation experiments is 5000.

We first present an experiment to obtain a reasonable threshold value in Definition 8. To obtain the threshold value δ , the function texts of all the cloud services are collected to form corpus. Then, Word2Vec is used to train the word vectors for the terms in the operation and theme. The value of δ is set as 0.7, 0.75, 0.8, 0.85, 0.9, and 0.95, respectively. For each value, we randomly select a cloud service as an invalid service. By computing function similarity and interface matching, we find substitutive one for it from 5000 cloud services. The accuracy and recall rating of substitution for different threshold values can be evaluated from Figure 4. By analyzing the trend of the two curves, we select 0.85 as the value of δ . The following experiments are conducted with δ set as 0.85.

Our method has two advantages. One is that the efficiency of service discovery in service substitution is improved by introducing service clusters and vector-based similarity calculation, and the other is that the recommended substitutive service is with a high collaboration similarity to the invalid one.

To verify performance of proposed method, we compare it with Santhanam et al. method [20], Sara et al. method [22], Wu et al. method [23], and Du et al. method [24]. Five rounds of experiments are performed in this study. These experiments in each round are performed for ten tests, and the average value of these results is taken as the final simulation result. According to different application areas, 5000 cloud services are manually divided into five parts. Different parts are selected to conduct experiments in turn. The number of cloud services, service clusters, and service nets of each round is shown in Table 1.

We make a rule that the component services in a service net can only be selected from different service clusters. That is, we cannot choose another cloud service from the same service cluster to compose a service net once we have chosen one from a service cluster. The number of cloud services in a service net is restricted within an interval of 8 to 20.

Algorithm execution time and service cooccurrence rate are compared between the above methods. As shown in Figure 5, our proposed algorithm has the least execution time in all rounds of experiments. Especially with increase in the number of cloud services, the advantage of our

Input: the cloud service cluster pool CSCP; the invalid cloud service Se;
Output: the substitutive cloud service St for Se.

```

(1) CSR =  $\emptyset$  cs_e =  $\emptyset$ ;
(2) for each Sec  $\in$  CSCP
(3)   compute FuncSim (Sec, Se)
(4)   if (Sec  $\leftrightarrow$  Se) CSR = CSR  $\cup$  {Sec};
(5) end for
(6) for each cs  $\in$  CSR.S
(7)   if (cs.I  $\propto$  Se.I  $\wedge$  Se.O  $\propto$  cs.O  $\wedge$  cs.Q  $\geq$  Se.Q)
(8)     cs_r = cs_r  $\cup$  {cs};
(9) End for
(10) For each cloud service S in cs_r
(11) RecomGrade (S) =  $\alpha$ *Qscore (S) +  $\beta$ *CollSim (S, Se);
(12) End for
(13) St = {S | max (RecomGrade (S))  $\wedge$  S  $\in$  cs_r};
(14) Return (St);

```

ALGORITHM 3: ServiceSubstitution (CSCP, Se).

FIGURE 4: Accuracy and recall rate for different threshold values.

TABLE 1: Component number for each round in experiment.

Round	Number of cloud services	Number of service clusters	Number of service nets
1	1000	80	300
2	2000	160	500
3	3000	240	800
4	4000	320	1000
5	5000	400	1200

algorithm's execution efficiency is more obvious. The result also shows that the algorithm execution time of clustering-based methods (our method, Du's method, and Wu's method) is lower than that of nonclustering method (Santhanam's method and Sara's method). Thus, we can get a conclusion that the clustering-based method can improve the efficiency of service replacement.

To prove that the substitutive services found by our method are more reasonable than other methods in process collaboration, we design another experiment to investigate service cooccurrence in service substitution. Let OccuNum (S_i) be the number of service S_i appearing in all the service

FIGURE 5: Comparison of algorithm execution time.

nets. Service cooccurrence of S_i and S_j is defined as $\text{ServiceCo_Occu}(S_i, S_j) = \text{OccuNum}(S_i \cap S_j) / (\text{OccuNum}(S_i) + \text{OccuNum}(S_j))$.

Service cooccurrence can be used to judge whether two cloud services are with a close collaborative relationship. If a substitutive one has a high service cooccurrence with precursor and successor of the invalid service, we think it is a

FIGURE 6: Comparison of service cooccurrence rate.

good collaboration-aware service substitution. Assume S_i , S_e , and S_j are three cloud services. Let S_i and S_j be precursor and successor of S_e , respectively. If S_e is not working and S_t is the substitutive service of S_e . Service cooccurrence in service substitution is defined as $\text{SubSerCo_Occu}(S_t, S_e) = (\text{ServiceCo_Occu}(S_i, S_t) + \text{ServiceCo_Occu}(S_j, S_t))/2$. From Figure 6, we can see that our method shows remarkably good performance in service cooccurrence. By numerical comparison in service cooccurrence rate, it is about 2 to 4 times higher than other methods. Thus, the proposed method integrates service collaboration well in the service substitution.

Experiments to show the efficiency in service discovery are also conducted. We compare our method with three recently proposed methods (Cheng et al. method [11], Zhang et al. method [12], and Nabli et al. method [15]). We have investigated two factors: service discovery time and top- k accuracy. Service discovery time reflects the search efficiency, while top- k accuracy is an illustration of discovery accuracy. Figure 7 shows the service discovery time in different round experiments. We can see that Nabli's method is the most efficient in all the methods. Our method got the second place, and its discovery time is nearly close with Nabli's method. Nabli's method is vector-based service discovery. All service vectors must be trained in advance. The existing service vectors are directly used to compute similarity; thus, it is with a high efficiency.

Compared with Nabli's method, service search speed of our method is slightly slow although we introduce service clusters and vector-based similarity calculation. The main reason is that we present interface matching in the service discovery.

In the experiment of top- k accuracy, we have revised data set and guaranteed that there are several groups of services which can be used to evaluate the discovery accuracy. Each group of services can respond to the same discovery requirement. The number of each service group is not less than k . In top- k experiment, we test the proportion of appropriate services in the first k services found by different methods. As shown in Figure 8, our method is with the highest accuracy in top- k service discovery experiments. However, Nabli's method is with

FIGURE 7: Comparison of service discovery time.

FIGURE 8: Comparison of service discovery accuracy.

the worst accuracy in all the methods. It is because Nabli's method is computed similarity based on the LDA topic model. The accuracy of the LDA topic model is greatly fluctuated by the service descriptive information.

6. Conclusions

To efficiently and reasonably find a substitutive cloud service for the invalid one, this work proposes a method to achieve service substitution. The searching space of finding the substitutive service is greatly reduced by introducing service clusters. To get the substitutive service, we first obtain a service candidate set by applying function similarity computing and parameters matching of service quality. Service collaboration is mined from the existing service processes. By comprehensive consideration of function, quality, and process collaboration, we propose an algorithm to achieve service substitution.

We innovatively obtain the similarity of service function and process collaboration by computing the cosine value of their word/service vectors. How to construct the vectors which can represent the feather of function similarity and collaboration intensity is discussed in detail in Section 4. Results of simulation experiments have shown that the proposed method significantly outperforms the state-of-the-art methods, especially for substitution in a mass of cloud services. In future work, we will focus on how to divide the process collaboration into different dimensions. A more

reasonable way to measure process collaboration will be presented so as to better realize service substitution.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Key R&D Program of China (Grant 2018YFB1702902), Natural Science Foundation of China (Grant 61973180), and Natural Science Foundation of Shandong Province (Grant ZR2019MF033).

References

- [1] B. Varghese and R. Buyya, "Next generation cloud computing: new trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.
- [2] <http://www.casicloud.com/>.
- [3] A. Vakili and N. J. Navimipour, "Comprehensive and systematic review of the service composition mechanisms in the cloud environments," *Journal of Network and Computer Applications*, vol. 81, pp. 24–36, 2017.
- [4] A. Jula, E. Sundararajan, and Z. Othman, "Cloud computing service composition: a systematic literature review," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3809–3824, 2014.
- [5] L. Sun, H. Dong, F. K. Hussain, O. K. Hussain, and E. Chang, "Cloud service selection: state-of-the-art and future research directions," *Journal of Network and Computer Applications*, vol. 45, pp. 134–150, 2014.
- [6] M. Parhi, B. K. Pattanayak, and M. R. Patra, "An ontology-based cloud infrastructure service discovery and selection system," *International Journal of Grid and Utility Computing*, vol. 9, no. 2, pp. 108–119, 2018.
- [7] M. M. Al-Sayed, H. A. Hassan, F. A. Omara et al., "An intelligent cloud service discovery framework," *Future Generation Computer Systems*, vol. 106, pp. 438–466, 2020.
- [8] Y. Xia, M. Zhou, X. Luo, S. Pang, and Q. Zhu, "Stochastic modeling and performance analysis of migration-enabled and error-prone clouds," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 2, pp. 495–504, 2015.
- [9] H. Mezni and M. Sellami, "Multi-cloud service composition using formal concept analysis," *Journal of Systems and Software*, vol. 134, pp. 138–152, 2017.
- [10] Lu. Zhao, W. Tan, N. Xie et al., "An optimal service selection approach for service-oriented business collaboration using crowd-based cooperative computing," *Applied Soft Computing*, vol. 92, Article ID 106270, 2020.
- [11] H. Cheng, M. Zhong, J. Wang et al., "Diversified keyword search based web service composition," *Journal of Systems and Software*, vol. 163, 2020.
- [12] F. Zhang, Q. Zeng, H. Duan, and C. Liu, "Composition context-based web services similarity measure," *IEEE Access*, vol. 7, pp. 65195–65206, 2019.
- [13] F. Chen, C. Lu, H. Wu, and M. Li, "A semantic similarity measure integrating multiple conceptual relationships for web service discovery," *Expert Systems With Applications*, vol. 67, pp. 19–31, 2017.
- [14] Y. Jiang, D. Tao, Y. Liu, J. Sun, and H. Ling, "Cloud service recommendation based on unstructured textual information," *Future Generation Computer Systems*, vol. 97, pp. 387–396, 2019.
- [15] H. Nabli, R. Ben Djemaa, I. A. Ben Amor et al., "Efficient cloud service discovery approach based on LDA topic modeling," *Journal of Systems and Software*, vol. 146, pp. 233–248, 2018.
- [16] I. Lizarralde, C. Mateos, A. Zunino, T. A. Majchrzak, and T.-M. Grønli, "Discovering web services in social web service repositories using deep variational autoencoders," *Information Processing & Management*, vol. 57, no. 4, p. 102231, 2020.
- [17] Y. Gong, L. Huang, and K. Han, "Service dynamic substitution approach based on cloud model," in *Proceedings of the First International Conference on Advanced Data and Information Engineering*, Springer, Berlin, Germany, 2014.
- [18] Q. Liang, B.-S. Lee, and P. C. K. Hung, "A rule-based approach for availability of service by automated service substitution," *Software: Practice and Experience*, vol. 44, no. 1, pp. 47–76, 2014.
- [19] M. W. Zhang, Z. L. Zhu, D. C. Li et al., "An execution context aware approach for Web service substitution," in *Proceedings of the 7th International Conference on Advanced Information Management and Service*, IEEE, Jeju, South Korea, November 2011.
- [20] G. R. Santhanam, S. Basu, and V. Honavar, "Web service substitution based on preferences over non-functional attributes," in *Proceedings of the IEEE International Conference on Services Computing*, IEEE, Chicago, IL, USA, September 2006.
- [21] H. Gao, Y. Duan, H. Miao, and Y. Yin, "An approach to data consistency checking for the dynamic replacement of service process," *IEEE Access*, vol. 5, pp. 11700–11711, 2017.
- [22] R. Sara, A. Bakhta, and L. Lakhdar, "A similarity network for web services operations substitution," *Journal of King Saud University-Computer and Information Sciences*, 2018, In press.
- [23] L. Wu, Y. Zhang, and Z. Di, "A Service-cluster Based approach to service substitution of web service composition," in *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design*, IEEE, Wuhan, China, May 2012.
- [24] Y. Du, J. Gai, and M. Zhou, "A Web service substitution method based on service cluster nets," *Enterprise Information Systems*, vol. 11, no. 10, pp. 1535–1551, 2017.
- [25] Q. Hu, Y. Du, and S. Yu, "Service net algebra based on logic Petri nets," *Information Sciences*, vol. 268, pp. 271–289, 2014.
- [26] Q. Hu, M. Liu, Z. Zhao, and J. Du, "A path detecting method to analyze the interactive compatibility of service processes based on WS-BPEL," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 19, p. e4699, 2018.
- [27] K. W. Church, "Emerging trends: a tribute to charles wayne," *Natural Language Engineering*, vol. 24, no. 1, pp. 155–160, 2017.
- [28] G. Huang, L. Zhou, X.-Z. Liu, H. Mei, and S.-C. Cheung, "Performance aware service pool in dependable service oriented architecture," *Journal of Computer Science and Technology*, vol. 21, no. 4, pp. 565–573, 2006.
- [29] W. Zai-jian, Y.-n. Dong, and X. Wang, "A dynamic service class mapping scheme for different QoS domains using flow aggregation," *IEEE Systems Journal*, vol. 9, no. 4, pp. 1299–1310, 2015.

- [30] Q. Hu, Z. Zhao, and J. Du, "A clustering method for isomorphic evolution of web services," *Scientific Programming*, vol. 2017, Article ID 5725864, 11 pages, 2017.
- [31] Q. Hu, Y. Y. Du, and P. Li, "Three-dimensional granularity division method for service clusters," *Journal of Software Engineering*, vol. 7, no. 4, pp. 133–141, 2013.