Information Sciences 433–434 (2018) 255–273

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

ISAT: An intelligent Web service selection approach for improving reliability via two-phase decisions

Weidong Wang[a,b,*], Zhangqin Huang[a,b,*], Liqiang Wang[c]

[a] Faculty of Information Technology, Beijing University of Technology, Beijing, China
[b] Beijing Engineering Research Center for IoT Software and Systems, Beijing University of Technology, Beijing, China
[c] Dept. of Computer Science, University of Central Florida, Orlando, FL, USA

ARTICLE INFO

Article history:
Received 3 March 2017
Revised 17 December 2017
Accepted 25 December 2017
Available online 26 December 2017

Keywords:
Web service
Optimization selection
Reliability
Decision approach

ABSTRACT

Due to stochasticity and uncertainty of malicious Web services over the Internet, it becomes difficult to select reliable services while meeting non-functional requirements in service-oriented systems. To avoid the unreliable real-world process of obtaining services, this paper proposes a novel service selection approach via two-phase decisions for enhancing the reliability of service-oriented systems. In the first-phase decision, we define the problem of finding reliable service candidates as a multiple criteria decision making (MCDM) problem. Then, we construct a decision model to address the problem. In the second-phase decision, we define the problem of selecting services based on non-functional requirements as an optimization problem. Finally, we propose a convex hull based approach for solving the optimization problem. Large-scale and real-world experiments are conducted to show the advantages of the proposed approach. The evaluation results confirm that our approach achieves higher success rate and less computation time to guarantee the reliability when compared to the other state-of-the-art approaches.

1. Introduction

Service- oriented architecture (SOA) patterns provide a flexible support for building software applications that use Web services available in typically large-scale and complex networks [32]. Web services are self-contained and loosely coupled reusable software components distributed and invoked over the Internet [31]. Compared with traditional stand-alone environments, the stochastic and unpredictable nature of open distributed environments based on SOA introduces new challenges in improving service-oriented software reliability [11]. Actually, the challenges are twofold. First, it is difficult to build fully reliable or fault-free service-based softwares under limited development cost and the pressure of time to market [28]. Then, it is uncertain whether users previously know which Web services are malicious. This is because the internal designs and implementation details of remote Web services from the third party are unclear to some extent [47].

In service-oriented software reliability researches [21], there are four possible approaches for improving reliability, which are fault prevention [36], fault removal [45], fault prediction [40], and fault tolerance [12]. Fault prevention and fault removal approaches need to revise the source code of Web services. They may be confined on the limited development cost and the pressure of time to market. Fault prediction approaches may result in inaccurate prediction because of the unpredictability of open distributed environments. Instead of removing faults or predicting faults, fault tolerance approaches

* Corresponding authors.
E-mail addresses: wangweidong@bjut.edu.cn, weidong_bjtu@126.com (W. Wang), zhuang@bjut.edu.cn (Z. Huang), lwang@cs.ucf.edu (L. Wang).

https://doi.org/10.1016/j.ins.2017.12.048
0020-0255/

have been widely applied in building reliable service-oriented softwares. Such approaches can be divided into three categories based on the roles in the SOA, *i.e.,* provider-, registry- and requester-based approaches, respectively. Provider-based approaches such as FT-Web [17] and FT-CORBA [15] mainly focus on developing functionally equivalent components running in parallel or operating another alternative backup component previously designed when the primary component crashes. Registry-based approaches such as an active UDDI (Universal Description, Discovery and Integration) [33] apply continuous access and discovery strategies to guarantee that users can obtain the available services needed. Requester-based approaches employ complex mediation strategies such as ws-reliability [24] to ensure the service to be successfully invoked by other services when their interface mismatches. Owing to the cost and time of developing redundant components in the provider-based approaches or designing complex fault tolerance strategies in registry- and requester-based approaches, the above three fault tolerance approaches are usually only used for critical softwares.

However, one of the promising fault tolerance approaches without paying much money and time for building reliable service-oriented softwares, commonly known as design diversity, is to adopt functionally equivalent although independently designed service candidates. The reason is that there are a large number of service candidates with equivalent function yet different QoS (Quality of Service) implemented by different organizations over the Internet and these service candidates can be employed as alternative components for tolerating faults. Complementary to previous fault tolerance approaches, which mainly focus on developing redundant components or designing complex fault tolerance strategies, this paper investigates how to optimally select fault tolerance components (services) to build reliable service-oriented softwares for the final goal of improving reliability.

Actually, the limitations of current service selection approaches are, they usually cannot find a good balance between speed and reliability: either fast and unreliable or slow and reliable. This is because, in an open distributed and service-oriented environment, there exist some unreliable Web services over the Internet that may be poor-quality, expensive, time-consuming, or even malicious. Furthermore, if these unreliable Web services cannot be well filtered, any effective service selection approach will become invalid since these unreliable Web services may result in QoS downgrade, or even lead to an ineffective selection process. Therefore, a filter for removing unreliable services is indispensable in supporting the selection process. In addition, as the number of service candidates increases, it may forbiddingly take large amount of time to obtain the optimal selection result according to the given QoS requirements proposed by users.

To address the issues discussed above, we present an intelligent Web service selection approach for improving reliability via two-phase decisions, namely ISAT. The ISAT considers not only how to avoid the selection of malicious services, but also how to maximize the QoS performance of Web services. Different from the previous version [38], the definition is extended to any number of non-functional constraints including linear and non-linear constraints. Also unlike the definition in [47], we typically consider the utility of a service as the only parameter of objective function since the other specific parameters, such as interest and error rate, are introduced in the first-phase decision process. Next, we summarize the major research contributions of this paper as follows.

- In the first-phase, we define the problem of identifying reliable service candidates as a multiple criteria decision making (MCDM) problem. Then, we propose the first-phase decision based on the technique for order of preference by similarity to ideal solution (TOPSIS) [10] to address this problem. Compared with traditional decision approaches, our customized decision typically considers multiple decision parameters from different dimensions (*e.g.,* user-, condition-, and environment-specific parameters) according to the characteristics of service-oriented systems.
- In the second-phase, we define the problem of selecting services for an optimal execution plan as the specific 0–1 integer programming problem. Then, we propose a convex hull based approach to solve efficiently this problem. Unlike traditional decision approaches, the convex hull based decision can reduce the search space of service candidates to the minimum while ensuring the success rate and accuracy of the solution.

Comprehensive experiments are conducted to study the success rate, computation time, and approximation ratio of our proposed approach compared with other competing approaches. The experimental results show the high success rate and efficiency of our approach. The rest of this paper is organized as follows. In Section 2, we introduce a motivating example to illustrate the research problem of the paper. In Section 3, we present the details of the two-phase decisions for Web service selection including the framework and QoS model of Web services. In Section 4, we give an illustrative example to show the computation process of the proposed approach. Section 5 discusses the experiment results. Section 6 reviews the state-of-the-art relevant to this work. Finally, Section 7 concludes this paper and outlines the future work.

2. Motivating example

We begin by a motivating example to illustrate the research problem. In this paper, an execution plan is an abstract description for the activities in a commodity trade process, which includes a group of tasks to execute according to a certain workflow. Fig. 1 indicates that the execution plan has six tasks and each task can execute by invoking a concrete component service. In the example, we assume that there are multiple functionally equivalent component Web service candidates provided by service communities. Meanwhile, we can use universal technologies to ensure the consistency of programming interface.

As the example in Fig. 1, there are several challenges to be addressed. (1) There are so many component Web services for the Task 1. We use Task 1 to make an order for the purpose of trade. Thus we need to identify which service candidates are

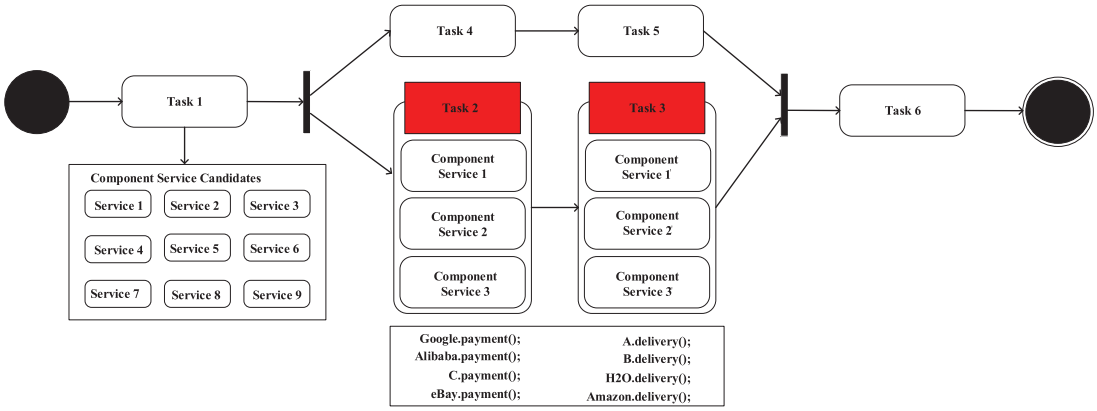

Fig. 1. An example of commodity trade scenario.

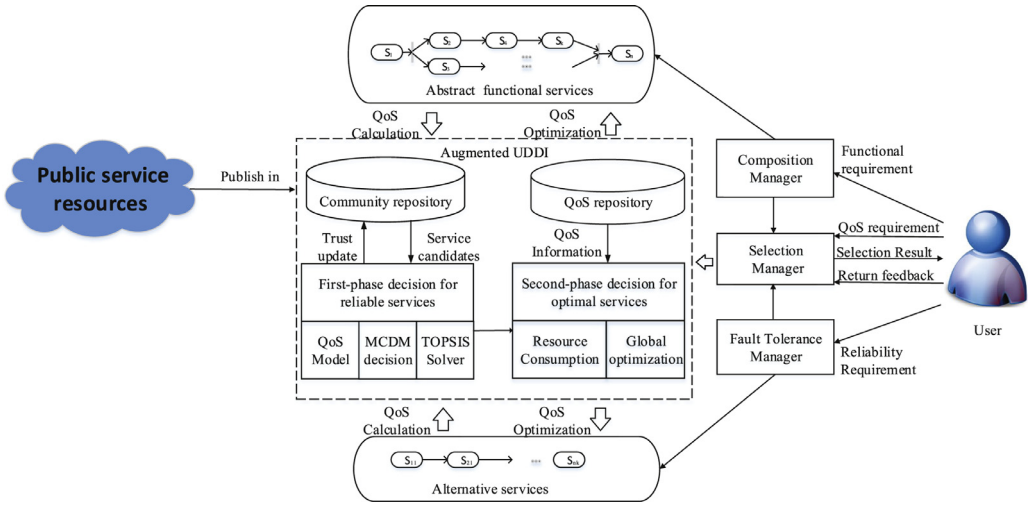

Fig. 2. The framework for efficient and reliable Web service selection via two-phase decisions.

reliable for Task 1 and which service candidates could be malicious or harmful for Task 1. (2) Tasks 2 and 3 are very critical jobs, payment and delivery, respectively. Once a fault occurs on Task 2 or Task 3, we need to select another alternative service to replace the failed one and find which candidate would be optimal while meeting global constraints. (3) For the purpose of maintenance, several services could be replaced and re-composed together for the maintenance requirement. So we need to know which service candidates will be selected to maximize overall QoS (utility) while meeting the given global constraints.

Therefore, we address these challenges by proposing a service selection approach via two-phase decisions, which defines decision model, identifies reliable services, and designs the corresponding selection algorithm to meet these challenges.

3. Two-phase decisions for Web service selection

In this section, we introduce the framework of two-phase decisions for Web service selection and the corresponding basic concepts. Then, we describe the QoS model of Web services for the two-phase decisions. Finally, we propose the two-phase decisions for Web service selection. In the first phase, the decision is made for identifying reliable service candidates. In the second phase, the decision is made by optimally selecting services. In this process, we design an objective function and use a convex hull based approach to find an almost optimal result. In the following sections, we will discuss the details of two-phase decisions.

3.1. Framework

In this section, we propose a framework for efficient and reliable service selection in composition or fault tolerance scenarios. Fig. 2 shows the framework. We provide in what follows the core components regarding the framework.

- Selection Manager (SM) optimally selects reliable services for the given execution plan, an abstract graph, generated by the composition manager or fault tolerance manager. For enhancing the performance of the SM, we employ two-phase decisions to select reliable services and find a near-optimal result in a reasonable short time. We will discuss the details in the following sections.
- Composition Manager (CM) considers the functional requests of users to generate an execution plan including a series of abstract tasks. An execution plan may have some work such as workflow process building, functional consistency checking, and interface matching. We will not discuss them in detail in this paper since it is beyond the scope of our work.
- Fault Tolerance Manager (FTM) provides the basis for a service provider to realize the delivery scheme and hence to offer fault tolerance as a service [22]. The FTM fulfils the target of transparently providing fault tolerance support and satisfies reliability goals. The FTM may have provided fault tolerance mechanisms such as fault detection and fault recovery. Hence, we will not discuss any details about these mechanisms, but focus on the problem that how to select and optimize alternative component services for a composite service.
- QoS repository is the container that collects a large number of Web services and their various measurements performed by commercial benchmark tools. In [29], the authors show some details about the QoS repository. In our QoS model, we investigate the generic QoS and specific domain QoS for Web services, then give an understandable description related with QoS attributes.
- Community repository is the container that collects Web services related to a specific area of interest. In [13], the authors show the information about the community repository. In our approach, we define interest by investigating users' feedbacks from the service community members and build the decision matrix associated with the interest. If these services have low interest value, they may be filtered out of the community.

The request of a user consists of the operations it invokes such as functional requirements and the QoS requirements including reliability. The functional requirements are implemented by matching functions from different communities published by the UDDI through the CM. The QoS requirements are achieved by the SM, which selects the optimal services according to the given execution plan generated by the CM. Meanwhile, the reliability requirements are schemed by the FTM that can decide which services need to be replaced by the alternative ones. Once the FTM generate s the execution plan related to fault tolerance, the SM will select the optimal services along the given fault tolerance route based on the plan.

3.2. Basic concepts

In this section, we introduce some preliminary information to formalize our approach. Hence, the basic concepts are listed as follows.

- Quality of service. Let $\mathrm{QoS}(s_{ij})=\{q_{ij}^1, q_{ij}^2, \ldots, q_{ij}^z\}$ represent a vector of generic quality attributes for service s_{ij}. z is the number of quality attributes. In this paper, the specific QoS attributes are given by their values associated to specific domain parameters.
- Global constraints. Let $ct = \{ct_1, ct_2, \ldots, ct_m\}$ be a vector of given global constraints in a composite service. Each ct_h denotes a constraint value provided by a user on the hth quality attribute.
- Utility function. Let $Utility(s_{ij})$ denote the utility function applied to s_{ij}, which indicates that the aggregation non-functional value achieved by s_{ij}.
- Execution plan. Let ET_l be an execution plan. It refers to a set of abstract tasks, which are related to the functional requests of users, and the relationships among these tasks.
- Composite service. Let $ST = \{ST_1, ST_2, \ldots ST_n\}$ be a composite service, where ST_i $(1 \le i \le n)$ is a abstract task corresponding to each function in the execution plan and integer n is the number of tasks required by the composite service.
- Alternative service. Let s'_{ij} be the alternative service. There are n' abstract tasks in ET_l to be bound by these alternative services according to the requirements of fault tolerance and composition.
- Malicious rate. Malicious service providers may offer malicious or inauthentic services. These services do not match what has been advertised in the WSDL (Web Services Description Language) document [27]. The malicious rate shows the share of malicious services in total number of services.

3.3. QoS model of Web services

In the process of service selection, we typically consider the quality parameters in the functionally equivalent Web services. Based on the investigations [6,46], we identify several representative parameters derived from QoS attributes. In [4,43], the authors basically use QoS models based on a set of quality parameters to solve the QoS-aware service selection problem. However, such models are not accurate enough while the Web services in different concrete tasks may have their own domain-specific quality parameters. Therefore, to describe every aspect of the non-functional characteristics of Web services, a more extendable and flexible QoS model is required by most QoS-aware service selection schemes. Therefore, in our QoS model, we divide QoS into two categories, generic QoS and specific QoS, respectively.

Table 1
Generic QoS description of Web service.

Parameter Name	Description
Response Time[1]	Time taken to send a request and receive a response (ms)
Latency[2]	Time taken for the server to process a given request (ms)
Availability[3]	Number of successful invocations/total invocations (%)
Throughput[4]	Number of invocations for a given period (invocations/s)
Successability[5]	Number of response messages/request messages (%)
Reliability[6]	Number of error messages/total messages (%)
Documentation[7]	Measure of documentation in WSDL (%)
Price[8]	Execution cost ($)

Table 2
Formulas for basic structures.

QoS	Basic structures			
z=	sequence	loop	branch	parallel
2,4,5,7	$\prod_{i=1}^{n} q_i^z$	$\sum_{i=0}^{n} x_i (q_1^z)^i$	$\sum_{i=1}^{n} x_i q_i^z$	$\prod_{i=1}^{n} q_i^z$
1,6	$\sum_{i=1}^{n} q_i^z$	$\sum_{i=0}^{n} x_i (q_1^z)^i$	$\sum_{i=1}^{n} x_i q_i^z$	$\arg\max_{i=1}^{n} q_i^z$
3,8	$\sum_{i=1}^{n} q_i^z$	$\sum_{i=0}^{n} x_i (q_1^z)^i$	$\sum_{i=1}^{n} x_i q_i^z$	$\sum_{i=1}^{n} q_i^z$

Table 3
An example of specific QoS for the audio service.

Parameter Name	Description
Interest	Users' attention for the audio service (%)
Error rate	The frequency of erroneous bits (%)
Service delay	The time elapsed while waiting for a response (ms)
Data rate	The rate in which data are encoded (Kbps)
Memory cost	The memory usage when running an application (Mb)
Bandwidth	The data transfer rate (Mbits/sec)

In Table 1, generic QoS is shared by each component service (e.g., response time and reliability). The values of QoS attributes are published by third-party agents, which obviously span diverse organizations and computing platforms. Accordingly, we identify several representative QoS parameters of Web services as follows.

In the above description, the generic QoS performance of a Web service can be represented as $q = (q^1, q^2, \ldots, q^8)$. Most of these parameters are derived from the public QoS dataset (*e.g.,* QWS dataset [1]). In addition, more QoS parameters can be easily added according to the users' requirements since the proposed QoS model is extensible. For composing service s and building their utility function, it is necessary to aggregate these above parameter values in different selection structures. Accordingly, we summarize formulas for four basic composite structures as follows.

Table 2 shows the formulas of calculating QoS values of composite services. In *loop* structure, x_i denotes a set of valid loops for i times, where n is the maximum loop times. In the *parallel* structure, the latency is the maximal value of n parallel branches. In the *branch* structure, x_i is the valid branch chosen, where $\sum_{i=1}^{n} x_i = 1$. Furthermore, there are some services from different domains, which require some specific QoS parameters to represent their QoS performance. Therefore, we introduce specific QoS parameters, namely specific QoS.

Specific QoS is derived from different specific domains. These Web services from different domains with different functionalities have different specific QoS. For instance, a audio service has QoS such as bandwidth and error rate, while a voice service has QoS such as clarity. Each specific QoS has its own criteria or rules to quantize their values [34]. In addition, it is necessary to introduce some specific QoS parameters related to users' subjective opinions such as interest since there exist some service providers who may deliver a low-quality service or malicious service, although they promise a high-quality service. Above all, the specific QoS can be employed for accurately identifying the characteristics of Web services from different specific domains. Table 3 illustrates an example of specific QoS for the audio service. We identify the typical specific QoS parameters such as error rate and interest for better descriptions of the audio service.

3.4. First-phase decision for preliminary selection

To describe the parameters related to reliable services, we consider user(φ-), condition(c-), environment(e-) parameters as decision parameters to construct decision matrix since these parameters cover most of the characteristics of services. Suppose that there are n abstract tasks belonging to the execution plan ET_l of a user as follows.

$$\{ST_1, ST_2, \ldots, ST_i\} \subseteq ET_l, \tag{1}$$

where ST_i denotes the ith task in the execution plan, $(0 \le i \le n)$.

First, we use φ parameter to measure a service whether it can be used for the given execution plan. To simply describe the φ parameter, we introduce usability and interest as the concrete user parameters. Different from the previous definition of usability, it commonly refers to the capability of a service of being understood by potential users such as the understanding of service interface complexity [8] or readability of its description (*e.g.,* WSDL) [30]. In this paper, the usability reflects the probability of the service s_{ij} to be selected by the yth user and it is defined in Eq. (2).

$$usability(s_{ij}) = \frac{\sum_{y=1}^{n_u} P(s_{ij}|u_y)}{n_u}, \tag{2}$$

where s_{ij} indicates the jth alternative service candidate in ST_i and $P(s_{ij}|u_y)$ denotes the conditional probability, which means the probability of the service s_{ij} to be selected by the yth user ($1 \le y \le n_u$). n_u denotes the number of users. Furthermore, to accurately define the interest of service, three hypotheses are proposed as follows. (1) The input user-service usage probabilities for all possible users and services are in the registry managed by the SM. (2) Some users could compromise applicability to the services with the same functionality and almost the same QoS performance. Hence, we ignore the differences of QoS performance in service selection. (3) We can obtain the quality rating results from two aspects as follows. For those services with the quality rating results by users' feedbacks, we directly use this quality rating results. For the services without quality rating by the given users, we could indirectly use a synthetic way to obtain the quality rating results by employing quality rating and ranking methods such as WsRF [5]. Thus the interest φ, which indicates a user's attention for the service s_{ij}, is defined in Eq. (3).

$$\varphi_{ij} = interest(s_{ij}) = \overline{P(s_{ij}|f_y)} \cdot usability(s_{ij}), \tag{3}$$

where $\overline{P(s_{ij}|f_y)} = \frac{\sum_{y=1}^{n_u} P(s_{ij}|f_y)}{n_u}$ $(y = 1, 2, \ldots, n_u)$. $P(s_{ij}|f_y)$ denotes the conditional probability that the positive feedback from the yth user occurs on the service s_{ij}.

Then, c parameter refers to the characteristics of service itself such as service's location. To make sense of the definition, we define the c parameter as follows.

$$c_{ij} = \{c_{ij}^1, c_{ij}^2, \ldots, c_{ij}^{n_c}\}, \tag{4}$$

where c_{ij} denotes n_c characteristics of the service s_{ij} such as service's price.

Finally, e parameter denotes the description of environment when services are invoked such as workload. The e parameter is defined as follows.

$$e_{ij} = \{e_{ij}^1, e_{ij}^2, \ldots, e_{ij}^{n_e}\}, \tag{5}$$

where e_{ij} denotes n_e environment parameters of the service s_{ij}. For instance, to obtain the θth environment parameter such as workload, we employ time window that covers the time when a service runs. The workload parameter e_{ij}^{θ} is defined as follows.

$$e_{ij}^{\theta} = \frac{\int_{t_a}^{t_b} e_{ij}^{\theta}(x)dx}{t_b - t_a}, \tag{6}$$

where $e_{ij}^{\theta}(x)$ denotes the distribution function of the θth e parameter ($1 \le \theta \le n_e$) with start time t_a and end time t_b. Actually, the workload parameter could be provided by the third-party organizations, who may test and publish the workload of service as specific domain QoS parameters for consideration.

To make an easy analysis, we transform the above parameters into a more structured and compact form. Hence, we use the decision matrix to describe the problem. Suppose that there are f alternative service candidates to be assessed based on the parameters mentioned above. Then, we incorporate all the parameters together and build the decision matrix as follows.

$$V(s_{ij}) = V(\varphi_{ij}, c_{ij}, e_{ij}) = (\varphi_{ij} c_{ij}^1 \ldots c_{ij}^{n_c} e_{ij}^1 \ldots e_{ij}^{n_e}) \tag{7}$$

where $V(s_{ij})$ is the decision vector (υ) of the service s_{ij} including parameters φ, c, and e. And each service s_{ij} has a decision vector υ. The decision matrix is composed of these vectors denoted as $M = (\upsilon_1, \upsilon_2, \ldots \upsilon_f)^T$. Furthermore, the problem of selecting reliable Web services is transformed into a problem of MCDM based on Eq. (8).

$$M = (\upsilon_1, \upsilon_2, \ldots \upsilon_f)^T = \begin{pmatrix} \upsilon_{11} & \ldots & \upsilon_{1g} \\ \vdots & \ddots & \vdots \\ \upsilon_{f1} & \ldots & \upsilon_{fg} \end{pmatrix}, \tag{8}$$

where g is the number of decision parameters for each service including φ, c, and e parameters. The dimensionality of M is dependent on g, where $g = 1 + n_c + n_e$. However, obtaining reliable service candidates may face the challenge of an increasing number of services that may contain some malicious services [37]. In the service computing research area, it is possible to efficiently solve the problem without artificially identifying these malicious services. Hence, we propose an improved technique for order preference by similarity to ideal solution to obtain reliable service candidates expected as follows.

Step 1: We normalize the decision matrix. The Euclidean Distance [25] based normalization method can be employed by considering the consistency (*i.e.,* the unified form of the following equations.) Thus we transform all the parameters existing in Eq. (8) by the normalization method shown in Eq. (9) as follows.

$$\bar{M} = (\overline{\upsilon_{ab}}) = \frac{\upsilon_{ab}}{\sqrt{\sum_{a=1}^{f} \upsilon_{ab}^2}} (a = 1, 2, \ldots, f, b = 1, 2, \ldots, g), \tag{9}$$

where $\overline{\upsilon_{ab}}$ is the normalized parameter. To build the decision criteria, we firstly find the best service and the worst service in the normalized matrix $\bar{M}$. Actually, these parameters that have a positive correlation with reliability are categorized in the set of positive parameters such as interest. Instead, these parameters that have a negative correlation with reliability are categorized in the set of negative parameters such as memory cost and error rate. To obtain the best service, we employ the ratio between the sum of values from the positive parameters and the sum of values from the negative parameters, where the best service may have the highest value of ratio.

Step 2: We find the best service and the worst service through the normalized decision matrix. The best service index can be found by Eq. (10) as follows.

$$best = \arg\max_{1 \le a \le f} \frac{\sum_{b \in B_+} \overline{\upsilon_{ab}}}{\sum_{b \in B-} \overline{\upsilon_{ab}}}, \tag{10}$$

where the *best* indicates the index of the best service, $B-$ and $B_+ \in \{1, 2, \ldots, g\}$. Meanwhile, $B-$ indicates the set of negative parameters. Instead, B_+ denotes the set of positive parameters. On the other hand, the worst service index can be found by Eq. (11) as follows.

$$worst = \arg\min_{1 \le a \le f} \frac{\sum_{b \in B_+} \overline{\upsilon_{ab}}}{\sum_{b \in B-} \overline{\upsilon_{ab}}}, \tag{11}$$

where the *worst* represents the index of the worst service.

Step 3: We calculate the distances in terms of the best service and the worst service. For each service in matrix $\bar{M}$, we calculate the ratio with the sum of values from the positive parameters and the sum of values from the negative parameters. Larger ratio indicates better performance than ath service. Similarly, lower ratio indicates worse service than ath service. Employing L^2-distance (Euclidean distance), the distance between the target alternative service and the best service can be calculated by Eq. (12) as follows.

$$d_{\upsilon_a \to \upsilon_{best}} = \sqrt{\sum_{b=1}^{g} (\overline{\upsilon_{ab}} - \overline{\upsilon_{best\,b}})^2}, (a = 1, 2, \ldots, f), \tag{12}$$

where $d_{\upsilon_a \to \upsilon_{best}}$ is the L^2-distance from the target alternative service to the best service. Meanwhile, the distance between the target alternative service and the worst service can be calculated as follows.

$$d_{\upsilon_a \to \upsilon_{worst}} = \sqrt{\sum_{b=1}^{g} (\overline{\upsilon_{ab}} - \overline{\upsilon_{worst\,b}})^2}, (a = 1, 2, \ldots, f), \tag{13}$$

where $d_{\upsilon_a \to \upsilon_{worst}}$ is the L^2-distance from the target alternative service to the worst service.

Step 4: We select k reliable services through similarity comparison. We define the similarity between the worst condition and the ath alternative service in Eq. (14) as follows.

$$sim(worst, a) = \frac{d_{\upsilon_a \to \upsilon_{worst}}}{d_{\upsilon_a \to \upsilon_{worst}} + d_{\upsilon_a \to \upsilon_{best}}}, (a = 1, 2, \ldots, f), \tag{14}$$

where the degree of $sim(worst,\ a)$ $(0 < sim(worst,\ a) < 1)$ indicates the value is closer to 1 meaning that the performance of the service is better. Specially, $sim(worst,\ a)$=1 indicates the alternative service has the best performance and $sim(worst,\ a)$=0 indicates the alternative service has the worst performance. Finally, top-k [35] alternative services are chosen according to the degree of $sim(worst,\ a)$. In addition, if there exist local constraints for any service, we could firstly remove the services that are unsatisfied with the local constraints in this step. Meanwhile, since service users may only provide a small number of local constraints, the unknown local constraints are set as $+\infty$ by default, so that all service candidates meet the constraints. Compared with traditional approaches [23,47], the proposed approach guarantees that the services obtained for further optimization selection are more reliable than the services obtained by traditional approaches. Above all, all the equations in the first-phase decision are designed by the TOPSIS and the experiments in the later section will prove its validation.

3.5. Second-phase decision for optimization selection

To obtain as many reliable services as possible, we design the first-phase decision in the previous subsection. Among these service candidates, we need further select optimal services for an execution plan while meeting the users' QoS requirements.

3.5.1. Decision problem definition

For a given execution plan ET_l, there exists a composite service $ST = \{ST_1, ST_2, \ldots, ST_i, \ldots, ST_n\}$ $(0 \le i \le n)$ and a given vector of global QoS constraints $ct = \{ct_1, ct_2, \ldots, ct_m\}$. The goal of the second-phase decision is to select a feasible service for each task ST_i while satisfying the global QoS constraints. We mathematically formulate the second-phase decision for optimization selection as follows.

Maximize:

$$utility(ET_l) \tag{15}$$

Subject to:

$$\forall l, \sum_{i \in ET_l} \sum_{j \in ST_i} q_{ij}^z x_{ij} \le ct_z (1 \le z \le h)^{\wedge} \tag{16}$$

$$\forall l, \prod_{i \in ET_l} \prod_{j \in ST_i} (q_{ij}^z)^{x_{ij}} \le ct_z (h < z \le m)^{\wedge} \tag{17}$$

$$\forall i, \sum_{j \in ST_i} x_{ij} = 1, \tag{18}$$

$$x_{ij} \in \{0, 1\}, \tag{19}$$

where n is the number of tasks, h is the number of linear constraints, m is the total number of constraints, and ct_z indicates the value of the zth non-functional constraint.

In this decision process, let Eq. (15) be the objective function and the relevant definition of utility is introduced in the later part of this section. Eq. (16) denotes the constraints for the linear attributes such as price and response time, where the aggregate attribute values of an execution plan are derived from the sum of all tasks within the execution plan. Eq. (17) represents the constraints for the attributes to be linearized such as availability, where the aggregate attribute values of an execution plan are the product of all tasks within the execution plan. In Eq. (18), x_{ij} is a decision variable. If x_{ij}=0, then $(q_{ij}^z)^{x_{ij}}$=1, showing that the service candidate is not selected, and vice versa. We employ Eqs. (18) and (19) to ensure that only one service candidate will be selected for each task within the execution plan. The value of x_{ij} is either 1 or 0, where x_{ij}=1 or $x_{ij} = 0$ indicates that a service candidate j is selected or not for the task i, respectively. The objective function and constraint functions should be linear in integer programming. Therefore, Eq. (17) needs to be transformed from non-linear to linear. Thus we use a linear equation as follows.

$$\forall l, \sum_{i \in ET_l} \sum_{j \in ST_i} x_{ij} ln(q_{ij}^z) \le ln(ct_z), (h < z \le m) \tag{20}$$

In order to evaluate the synthetical quality of services within an execution plan, we define the synthetical utility function as follows.

$$utility(ET_l) = \sum_{ST_i \in ET_l} utility(ST_i) = \sum_{s'_{ij} \in ST_i} utility(s'_{ij}), \tag{21}$$

where s'_{ij} is a service candidate selected by the first-phase decision in the execution plan ET_l. In order to evaluate the multi-dimensions quality of each given Web service, we define the utility [42] of each service in Eq. (22) as follows.

$$utility(s'_{ij}) = \sum_{\alpha=1}^{n'} \omega_\alpha \times \left(\frac{q_{ij}^\alpha - \mu_i^\alpha}{\sigma_i^\alpha} \right) + \sum_{\beta=n'+1}^{m} \omega_\beta \times \left(1 - \frac{q_{ij}^\beta - \mu_i^\beta}{\sigma_i^\beta} \right), \tag{22}$$

where ω_α is the weight corresponding to QoS attributes to be maximized and ω_β is the weight corresponding to QoS attributes to be minimized $(0 < \omega_\alpha, \omega_\beta < 1)$, $\sum_{\alpha=1}^{n'} \omega_\alpha + \sum_{\beta=n'+1}^{m} \omega_\beta = 1$. μ and σ are the average and standard deviation of utility values for all candidates in the same task. In the definition of utility function, all attributes have user-defined weights by their importance. The utility function definition is based on their averages and standard deviations to ensure that the function will not be biased by any attribute with a large value.

In this way, we formulate the decision of optimization selection as a 0–1 integer programming problem. The integer programming problem is known as NP-Complete problem. And the problem can be solved by exhaustive search. However, as the size of the problem grows, it takes forbiddingly large amount of time to find the optimal solution. For the sake of computation speed, in [23,47], the authors employed some heuristic algorithms to solve the problem. To further enhance the success rate of the algorithm, we propose a convex hull based approach to find an optimal solution.

3.5.2. Convex hull based solution

The basic idea of the convex hull based solution is to find an optimal solution by combining convex hulls in the 2-dimensional space for the reduction of search space. It is significantly reduced while still promising that the obtained result is almost optimal. The solution includes the following steps.

Step 1: We map service candidates to convex hulls. First, we build the coordinate of convex hulls, where the vertical axis represents the utility value of a service and the horizontal axis represents the resource consumption value of a service calculated as follows.

$$resc(s'_{ij}) = \sum_{\gamma=1}^{\delta} \omega_\gamma \times (q_{ij}^{\gamma} - \mu_i^{\gamma}), \tag{23}$$

where resc(s'_{ij}) indicates the resource consumption function of the service s'_{ij}. Let μ_i^{γ} be an average consumption value of the ith task and δ is the number of consumption attributes. Then, each service can be mapped to a point in the 2-dimensional space by calculating its resource consumption value and utility value. Finally, the convex hulls are constructed for each task by employing the graham-scan algorithm [20].

Step 2: We identify dominant services within the convex hulls. These dominant services play a crucial role in the reduction of the search space since they may defeat other non-dominant services ignored in the selection process. Each dominant service related to our approach needs to be customized accordingly, and can be derived from the rules defined as follows.

Rule 3.1. The ρth service is dominated by a single service ϱ, written as $\rho \ll_m \varrho$. Formally, $\exists k$, if $resc(s'_{i\rho}) \geq resc(s'_{i\varrho})$ and $utility(s'_{i\rho}) \leq utility(s'_{i\varrho})$ then $\rho \ll_m \varrho$.

Rule 3.1 indicates the service with high utility value but requiring low-aggregate resource consumption should be selected first. For example, for the same task i in which there are two service candidates, the resource consumption of service ρ is higher than the service ϱ and the utility of service ρ is lower than the service ϱ. Hence, we have reason to believe that the service ϱ would to be chosen in the same condition.

Rule 3.2. A service set H is dominated by the ρth service, written as $H \ll \rho$. Formally, $\forall \varrho \in H$, if $resc(s'_{i\rho}) \leq resc(s'_{i\varrho})$ and $utility(s'_{i\rho}) \geq utility(s'_{i\varrho})$ then $H \ll \rho$.

Rule 3.2 describes a group of services with high resource consumption and low utility are ignored in selection process since the service with low resource consumption and high utility exists in the same task. For example, for the same task i in which there are a group of service candidates, the resource consumption of service ρ is lower than the services in the set H and the utility of service ρ is higher than the services in the set H. Hence, the service ρ should be firstly considered in the same condition.

Rule 3.3. The ρth service is dominated by a service ϱ plus ε, written as $\rho \ll_{\equiv} \varrho$, where ε is the best service, *i.e.,* $\forall\ \rho$, $\exists\ \frac{utility(s'_{i\rho})}{resc(s'_{i\rho})} \leq \frac{utility(s'_{i\varepsilon})}{resc(s'_{i\varepsilon})}$. Formally, if $\frac{utility(s'_{i\varepsilon})+utility(s'_{i\varrho})}{resc(s'_{i\varepsilon})+resc(s'_{i\varrho})} \geq \frac{utility(s'_{i\rho})}{resc(s'_{i\rho})}$ then $\rho \ll_{\equiv} \varrho$.

Rule 3.3 denotes the service that provides larger utility value per unit of aggregate resource consumption should be preferentially selected in the same condition. For example, if a service ϱ around the best service ε has a larger increase on the utility per unit of aggregate resource consumption than the service ρ. Then the service ϱ, namely a dominated service against the service ρ, should be chosen in an execution plan.

Step 3: Service candidates mapped as dominant services are mainly participated in selection and optimization process. We employ the integer programming solver to select an almost optimal solution among the dominant services since the final result will most likely be produced among these dominant services. Above all, the convex hull based approach can remove non-dominate services and retain dominate services to reduce the search space. Moreover, the almost optimal solution could derive from these dominate services. Indeed, it results in a suboptimal solution yet such a solution can achieve high approximation ratio and success rate, which will be proved by the following understandable experiments in Section 5.

3.6. Computational complexity analysis

In this section, we calculate the computational complexity of ISAT as follows. For the first-phase decision, matrix normalization requires O(fg) operations. The steps 1–4 requires O(fg) operations. The final top-k algorithm requires O($f \cdot lnk$) operations. Hence, the computational complexity of the first-phase decision is O(fg)+O(fg)+O($f \cdot lnk$) = O(fg).

For the second-phase decision, in the step 1, the resource computation requires O(nl) operations and the convex hull construction requires O($nl \cdot log_2 n$) operations, where n is the number of tasks and l is the number of services in each task. In the step 2, dominant service identification requires O(nl) operations. In the step 3, the worst-case complexity for the final integer programming is O($n^2 l' m$). Note that l' is the number of dominated services, $f \gg l \gg l'$. Therefore, the computational complexity of the second-phase decision is O($nl \cdot log_2 n$)+O(nl)+O($n^2 l' m$)=O($n^2 l' m$).

Above all, the total computational complexity of the two-phase decisions is O(fg)+O($n^2 l' m$) = O($n^2 l' m$). As will be shown in the experiments in Section 5, our approach provides higher success rate and less computation time than traditional approaches even in malicious environments.

Table 4
Specific domain QoS of voice communication services.

Function	Service	RC	IN	ER	SD	DR	MC	BW
Voice communication	s_{11}	0.49	0.25	0.03	50	128	128	20
	s_{12}	0.42	0.75	0.09	100	56	256	4
	s_{13}	0.66	0.34	0.10	70	55	512	8
	s_{14}	0.73	0.09	0.11	150	76	1024	1
	s_{15}	0.32	0.46	0.08	155	85	512	4
	s_{16}	0.64	0.66	0.06	60	90	512	10
	s_{17}	0.16	0.02	0.11	160	100	512	2

4. An illustrative example

For better understanding of the ISAT approach proposed, we use an example to illustrate the computation process of the approach. Here, we choose an online radio broadcasting scenario as the example because in which multiple subscribers could simultaneously employ the voice communication service to receive voice streams with various quality specified.

In our example, we employed the design method of the experiment on voice communication services [41] to obtain the QoS information, which was collected from service providers and subscriber(s). In the example, we used three service parameters to generate different QoS performance as follows. The sampling rate (e.g. 44.1 KHz, 88.2 KHz, 132.3 KHz, 176.4 KHz, and 220.5 KHz) was applied for recording the chosen stream. The number of subscribers was introduced to reflect the service's usage frequency. The size of buffer (e.g. 16 KB, 24 KB, 32 KB, 40 KB, and 48 KB) was employed for storing the voice stream before network transmissions. We recorded 30 data observations during the period of one minute. Then, we collected the feedback among different subscribers to calculate the user-service interest probability.

In Table 4, RC stands for resource consumption, IN stands for interest, ER stands for error rate, SD stands for service delay (ms), DR stands for data rate, MC stands for memory cost, and BW stands for bandwidth (Mbits/sec). The table includes the specific QoS (*e.g.,* memory cost). For instance, online radio broadcasting requires the service delay to be less than 200 ms, data rate larger than 56 Kbps, and error rate less than 0.15. In the online broadcasting scenario, there may exist many users with different minimum data quality requirements. Therefore, under given limited system resources, it is necessary to adapt the configuration (design diversity) of online radio broadcasting to provide massive audio data with various quality to satisfy different users' specific QoS requirements while serving more users with improved audio quality.

Basically, we use the first-phase decision to identify the reliable services. Accordingly, we buil d the matrix M based on the data from Table 4 as follows.

$$M = \begin{pmatrix} 0.25 & 0.03 & 50 & 128 & 128 & 20 \\ 0.75 & 0.09 & 100 & 56 & 256 & 4 \\ 0.34 & 0.10 & 70 & 55 & 512 & 8 \\ 0.09 & 0.11 & 150 & 76 & 1024 & 1 \\ 0.46 & 0.08 & 155 & 85 & 512 & 4 \\ 0.66 & 0.06 & 60 & 90 & 512 & 10 \\ 0.02 & 0.11 & 160 & 100 & 512 & 2 \end{pmatrix}, \tag{24}$$

where $M = (\upsilon_1, \upsilon_2, \upsilon_3, \upsilon_4, \upsilon_5, \upsilon_6)^T$ and υ_i=($\langle IN\rangle$, $\langle ER\rangle$, $\langle SD\rangle$, $\langle DR\rangle$, $\langle MC\rangle$, $\langle BW\rangle$), $(1 \le i \le 6)$.

Step1: According to Eq. (9), we have normalized all the QoS parameters in Eq. (25). The normalized decision matrix is as follows.

$$\bar{M} = \begin{pmatrix} 0.999 & 0.263 & 0.307 & 0.788 & 0.242 & 4.472 \\ 2.998 & 0.789 & 0.615 & 0.344 & 0.485 & 0.894 \\ 1.359 & 0.877 & 0.430 & 0.338 & 0.970 & 1.788 \\ 0.358 & 0.964 & 0.923 & 0.467 & 1.940 & 0.223 \\ 1.834 & 0.701 & 0.954 & 0.523 & 0.970 & 0.894 \\ 2.631 & 0.526 & 0.369 & 0.554 & 0.970 & 2.236 \\ 0.079 & 0.964 & 0.985 & 0.615 & 0.970 & 0.447 \end{pmatrix}, \tag{25}$$

where $\bar{M}$ is the normalized decision matrix.

Step2: We find the best service index and the worst service index using Eqs. (10) and (11) as follows.

$$\begin{cases} best = \arg\max_{a=1} \dfrac{\sum_{b\in B_+} (\overline{\upsilon_{11}} + \overline{\upsilon_{14}} + \overline{\upsilon_{16}})}{\sum_{b\in B-} (\overline{\upsilon_{12}} + \overline{\upsilon_{13}} + \overline{\upsilon_{15}})} = 7.691 \\ worst = \arg\min_{a=4} \dfrac{\sum_{b\in B_+} (\overline{\upsilon_{41}} + \overline{\upsilon_{44}} + \overline{\upsilon_{46}})}{\sum_{b\in B-} (\overline{\upsilon_{42}} + \overline{\upsilon_{43}} + \overline{\upsilon_{45}})} = 0.274, \end{cases} \tag{26}$$

where the group of parameters $\langle IN, DR, BW\rangle$ belongs to positive parameters (B+) and $\langle ER, SD, MC\rangle$ belongs to negative parameters (B-). These results indicate that s_{11} is the best service (a = 1) and s_{14} is the worst service (a = 4).

Table 5
Generic QoS of voice communication services.

S	Name	RP	LA	AV	TH	SU	RE	DO	PR
s_{11}	NumPager	151.75	69.10	88.0	8.7	96.5	73.0	7.0	12.0
s_{12}	SMS	106.89	28.14	95.1	5.5	99.4	73.0	1.0	45.0
s_{13}	Phone	98.09	21.54	87.3	17.2	96.2	67.0	28.2	37.0
s_{15}	File	112.83	45.16	98.2	4.1	100.1	73.0	16.3	56.0
s_{16}	Personalizer	426.17	3.50	91.8	6.3	97.5	73.0	12.0	67.0

Step3: We calculate the distances according to the best service and the worst service. For example, we calculate the distances from s_{12} to the best service s_{11} and the worst service s_{14} as follows.

$$\begin{cases} d_{s_{12}\to s_{11}} = \sqrt{\sum_{b=1}^{6}(\overline{v_{2b}} - \overline{v_{1b}})^2} = 4.171 \\ d_{s_{12}\to s_{14}} = \sqrt{\sum_{b=1}^{6}(\overline{v_{2b}} - \overline{v_{4b}})^2} = 3.103, \end{cases} \tag{27}$$

where $d_{s_{12}\to s_{11}}$ and $d_{s_{12}\to s_{14}}$ are the L^2-distance from the target alternative service s_{12} to the best service s_{11} and the worst service s_{14}, respectively. In the same way, we can obtain $d_{s_{13}\to s_{11}} = 2.907$, $d_{s_{14}\to s_{11}} = 4.723$, $d_{s_{15}\to s_{11}} = 3.835$, $d_{s_{16}\to s_{11}} = 2.508$, $d_{s_{17}\to s_{11}} = 4.307$, $d_{s_{13}\to s_{14}} = 2.156$, $d_{s_{15}\to s_{14}} = 1.908$, $d_{s_{16}\to s_{14}} = 3.265$, and $d_{s_{17}\to s_{14}} = 1.046$.

Step4: We calculate the similarity between the worst service s_{14} and the ath alternative service ($a = 1, 2, \ldots, 7$) using Eq. (14) as follows.

$$\begin{cases} sim(s_{14}, s_{11}) = \dfrac{d_{s_{11}\to s_{14}}}{d_{s_{11}\to s_{14}} + d_{s_{11}\to s_{11}}} = 1.000, \\ sim(s_{14}, s_{12}) = \dfrac{d_{s_{12}\to s_{14}}}{d_{s_{12}\to s_{14}} + d_{s_{12}\to s_{11}}} = 0.426, \\ sim(s_{14}, s_{13}) = \dfrac{d_{s_{13}\to s_{14}}}{d_{s_{13}\to s_{14}} + d_{s_{13}\to s_{11}}} = 0.425, \\ sim(s_{14}, s_{14}) = \dfrac{d_{s_{14}\to s_{14}}}{d_{s_{14}\to s_{14}} + d_{s_{14}\to s_{11}}} = 0, \\ sim(s_{14}, s_{15}) = \dfrac{d_{s_{15}\to s_{14}}}{d_{s_{15}\to s_{14}} + d_{s_{15}\to s_{11}}} = 0.332, \\ sim(s_{14}, s_{16}) = \dfrac{d_{s_{16}\to s_{14}}}{d_{s_{16}\to s_{14}} + d_{s_{16}\to s_{11}}} = 0.565, \\ sim(s_{14}, s_{17}) = \dfrac{d_{s_{17}\to s_{14}}}{d_{s_{17}\to s_{14}} + d_{s_{17}\to s_{11}}} = 0.195, \end{cases} \tag{28}$$

where top-k alternative services are chosen according to the degree of $sim(s_{14}, a)$ in descending order, *i.e.,* s_{11}, s_{16}, s_{12}, s_{13}, s_{15}. Actually, it is not difficult to find that the discarded services, s_{14} and s_{17}, have poor QoS attributes such as interest and bandwidth. In this way, we can efficiently select specific number of services with better QoS performance including reliability.

Then, we apply the second-phase decision to optimize a composite service. Commonly, one or more atomic services could be replaced by the alternative service(s) with equivalent function yet different QoS in the given composite service according to the user's requirement *e.g.,* fault tolerance or maintenance. In this process, the users may consider more about generic QoS parameters since they are not good at specific domain QoS parameters with the service(s) to be updated. The corresponding generic QoS parameters are as follows.

In Table 5, RP stands for response time, LA stands for latency, AV stands for availability, TH stands for throughput, SU stands for successability, RE stands for reliability, DO stands for documentation, and PR stands for price. We calculate the utility for each service listed in the table using Eq. (22). For instance, the computation process of the utility of s_{12} is as follows. The empirical weight value can be set as ⟨RP:0.10, LA:0.05, AV:0.05, TH:0.05, SU:0.30, RE:0.30, DO:0.05, PR:0.10⟩.

$$utility(s_{12}) = \sum_{j=1}^{2}\omega_j \times \left(1 - \frac{q_{2j} - \mu_j}{\sigma_j}\right) + \sum_{j=3}^{8}\omega_j \times \left(\frac{q_{2j} - \mu_j}{\sigma_j}\right) = 0.53 \tag{29}$$

In the same way, we also obtain $utility(s_{11}) = -0.24$, $utility(s_{13}) = -0.49$, $utility(s_{15}) = 0.80$, $utility(s_{16}) = 0.14$. Note that the minus "−" indicates its utility below the average. In this example, under the given resource consumption in Table 4, we draw the following graph to build the relation between utility and resource consumption, where the horizontal axis represents the utility and the vertical axis indicates the resource consumption. Next we map each service on it as shown in Fig. 3.

According to Rule 3.1, the service s_{12} is dominated by the single service s_{15} since $utility(s_{12}) < utility(s_{15})$ and $rec(s_{12}) > rec(s_{15})$. For the other points s_{11}, s_{12}, s_{13}, and s_{16}, we use the dash line to connect them then obtain the quadrilateral ⟨s_{11}, s_{12}, s_{13}, s_{16}⟩. Moreover, in terms of Rule 3.2, the service set in the quadrilateral ⟨s_{11}, s_{12}, s_{13}, s_{16}⟩ is dominated by

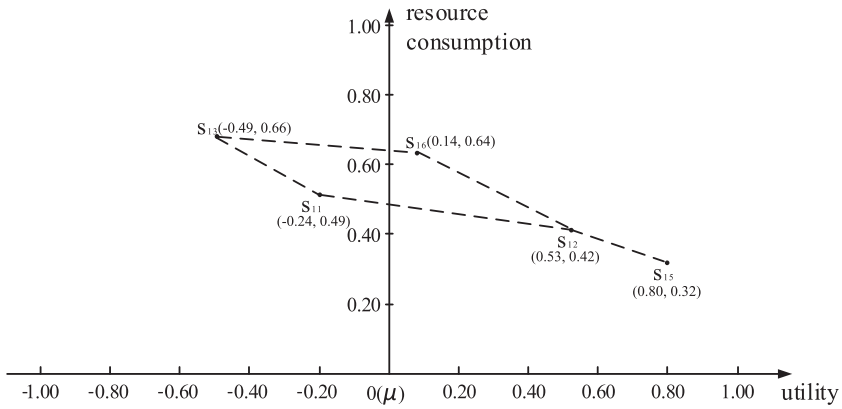

Fig. 3. An example of convex hull.

the service s_{15}. s_{15} has a higher utility value with lower resource consumption than the other services in the quadrilateral. If s_{15} is not chosen in the optimal solution, all the services in the quadrilateral may be discarded in the global optimization process. The experiments in the following section will demonstrate that a large amount of computation time can be saved although the optimality may be a very small loss.

5. Experiments

5.1. Experiment setup

Data source. To evaluate different decision model aspects, the publicly available QWS dataset [1] was used for the experiments. To obtain non-functional property values of Web services, we directly access the Web portal of QWS Dataset, which comprises measurements of 9 non-functional attributes for 2507 typical real-world Web services for Web service researchers. Since the QWS dataset does not provide quality rating (user-service interest) information used to perform the experiments, we indirectly used a WsRF [5] based synthetic way to simulate the quality rating. In order to simulate the value as accurately as possible in the real world, we generated a random value as user-service interest probability by considering the overall quality rating or QoS performance of a service. In the QWS Dataset, there are 4 different quality rating levels or QoS performance classifications provided by the WsRF. Hence, we could categorize the range of user-service interest probability into 4 ranges [0,0.25), [0.25,0.50), [0.50,0.75), and [0.75,1], respectively, which were corresponding to 4 classifications. For example, we can generate a random value (0.78) belonging to the range [0.75,1] as user-service interest probability for the service "FastWeather" in the best QoS performance classification invoked by the user A, and a random value (0.89) by the user B.

To further ensure that the results of the experiments are not biased by the QWS database used, another dataset with a larger number of services was created by employing the DTM Data Generator[1] by simulating non-functional attributes in Web service applications.

Evaluation metric. To measure the accuracy of the proposed approach, we apply the approximation ratio [26] defined as follows.

$$Approximation\ ratio = \frac{U_{obtained}}{U_{optimal}}, \tag{30}$$

where $U_{obtained}$ denotes the total utility value of Web services obtained, and $U_{optimal}$ denotes the total utility value of the optimal selection result. As the motivating example illustrated in Section 2, the third-party payment service in an execution plan could not work well or crash while a large number of users are accessing simultaneously. The reasons of invocation failures may be summarized as follows. (1) The delivery QoS metric(s) such as the throughput could not satisfy what it promises, thus such a payment service may be considered as a malicious service. (2) The throughput is greatly beyond its limited throughput specified. That the workload dramatically increases may be another reason for invocation failure. To evaluate the success ratio metric of each execution plan, we employ the success rate [23] defined as follows.

$$Success\ rate = \frac{N_{success}}{N'}, \tag{31}$$

where $N_{success}$ denotes the number of execution plans correctly selected, and N' denotes the total number of execution plans. In addition, we conducted the experiments on HP 8280 with four Intel Cores i5-2400 CPU of 3.1 GHz and with 8GB RAM. The ISAT is implemented by Java 6.0 and runs on Windows 7.

[1] http://www.sqledit.com/dg/ .

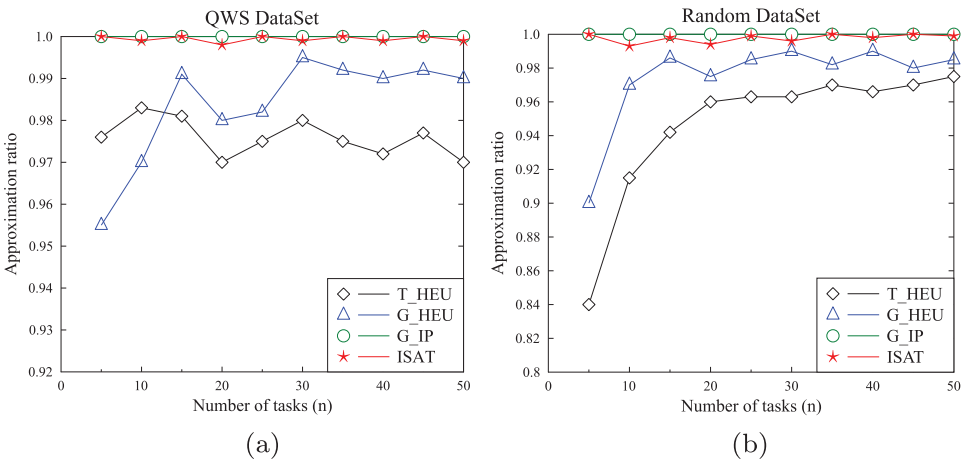

Fig. 4. Performance on approximation ratio using different approaches.

5.2. Performance comparison

To study the decision performance, we compare ISAT proposed with three other decision algorithms including T_HEU [23], G_HEU [47], and G_IP [38]. The reasons why these particular competitors are chosen can be summarized as follows. First, the G_IP returns the global optimization solution, therefore it is used as a baseline in the experiments. As the proposed ISAT approach is a near-optimal one, the quality of the results obtained by ISAT needs to be evaluated by comparing them with optimal results by the global optimization approaches, such as the G_IP. Then, T_HEU was proposed as a heuristic approach by considering reliability for service selection in the process of QoS-aware composition, which can achieve high success rate and fast speed while meeting the given global QoS constraints. Finally, G_HEU, as one of the heuristic optimal approaches in the fault tolerance service selection research, can obtain a solution with high accuracy, considering not only single metric (*i.e.,* reliability), but also a number of other QoS properties (*e.g.,* response time and throughput). By taking the advantages of these approaches, we choose them as the particular competitors. Some details about these approaches are as follows.

- Heuristic optimization approach based on global constraints(T_HEU) [23] employs a heuristic optimization approach by combining with a filter component and convex hull based search space deduction.
- Local and global constraints based heuristic approach (G_HEU) [47] formulates the problem of fault tolerant Web service selection based on the local and global constraints as an optimization problem and designs a local and global constraints based heuristic approach.
- Fault tolerant Web service selection framework (G_IP) [38] employs a resilient framework to automatically select fault tolerant Web services by an integer programmer (IP) based on global constraints for failed services in a workflow.
- Two-phase decisions approach(ISAT) employs the two-phase decisions to select fault tolerant Web services based on global constraints for enhancing the reliability. The details are shown in the previous Section 3 in this paper.

Fig. 4 shows the approximation ratio using different approaches including heuristic algorithms and integer programming algorithms by varying number of tasks from 5 to 50. The number of services in each task was fixed at 100 and the number of malicious services was fixed at 0 respectively. For T_HEU and G_HEU, the maximal approximation ratio are 98.5% and 99.6% in both QWS dataset and random dataset, separately, while the approximation ratio of G IP and ISAT are almost 1, since they are global optimization approaches. The results show that ISAT performs well in both QWS dataset and random dataset.

Fig. 5 shows the computation time achieved by the different approaches including heuristic algorithms and integer programming algorithms by changing the number of tasks from 5 to 50. The number of services in each task was fixed at 100 and the number of malicious services was fixed at 0 respectively. In the experiment, we observe that the ISAT performs well for both QWS dataset and random dataset. The computation time of ISAT is below 100 ms by varying the number of tasks from 0 to 30. These values are much better than those achieved by G_IP. However, when the number of tasks is beyond 30, the computation time of ISAT slowly increases as the number of tasks increases. Meanwhile, we also observe that both T_HEU and G_HEU demand less computation time than ISAT because they may sacrifice partial accuracy for the sake of deduction on computation time. Accurately, for fault tolerant services in service computing systems, it is reasonable that the accuracy is critical parameter rather than the computation time especially in these scenarios, where the computation time achieved by different approaches are almost the same. To investigate how the decision parameters impact the results of an execution plan, we conduct the following group of tests in different scenarios.

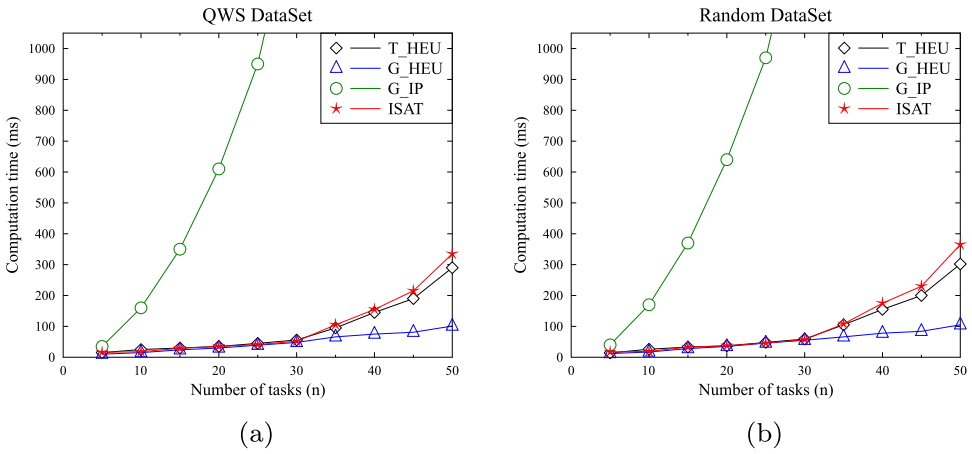

Fig. 5. Performance on computation time using different approaches.

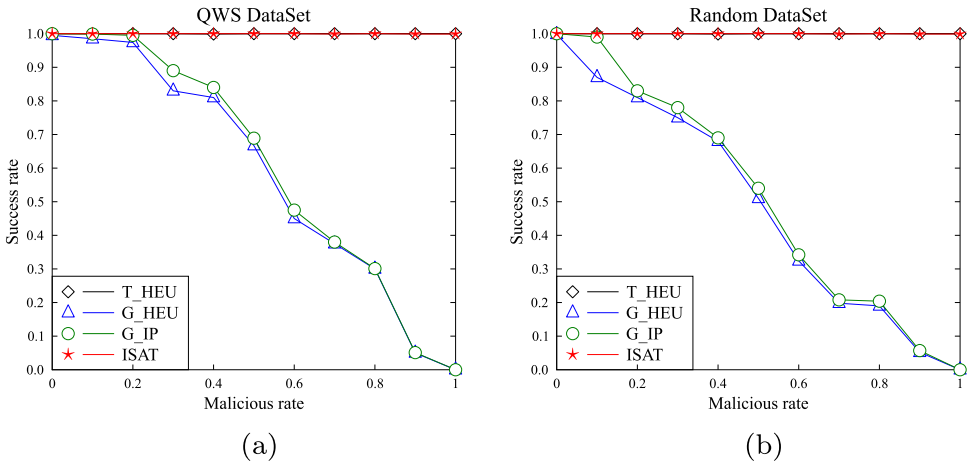

Fig. 6. Performance on success rate using different approaches.

Fig. 6 shows the comparison among the success rates obtained by the different approaches with different malicious rates of services. The results indicate that the success rate of T_HEU and ISAT remains almost 1, while other approaches dramatically drop as the malicious rate increases. This is because an increasing number of malicious services may be selected by the other approaches without considering the unreliable services. The approach ISAT benefits from the first-phase decision process that filters out all the unreliable services for the second-phase decision making.

5.3. Impact of decision parameters

The experimental results show that the proposed approach is valid and efficient. However, it is unclear whether the first-phase algorithm has great impact on the performance. In order to study the impact of the first-phase decision, we conduct two implementations of ISAT. One implementation employs the first-phase decision, while the other does not. In the experiment, we set $k = 10$, $c = 10^3$, $\varphi = 0.35$, $e = 10^3$, and given number of tasks $= 50$. Meanwhile, the number of services in each task was fixed at 100. We test success rate (SR) and approximate ratio (AR) at 4 different malicious rates, 10%, 25%, 40% and 70%, separately. Table 6 shows the ISAT with the first-phase decision achieves significant enhancement of success rate. For example, when the malicious rate is 25%, the ISAT with the first-phase decision achieves success rate (1) higher than the success rate (0.795) without the first-phase decision. The higher success rate of the approach with the first-phase decision is the direct result from its filtering process. Table 6 also shows the ISAT with the first-phase decision always obtains almost the same result on approximation ratio as the approach without the first-phase decision. The above experimental results indicate that the first-phase decision has little effect on the approximation ratio obtained and could be ignored as discussed above.

Table 6
Impact of the first-phase decision.

Malicious rate	Without the first decision		With the first-phase decision	
	SR	AR	SR	AR
10%	0.854	0.996	0.998	0.995
25%	0.795	0.999	1.000	0.998
40%	0.702	0.997	0.999	0.997
70%	0.458	0.997	0.995	0.996

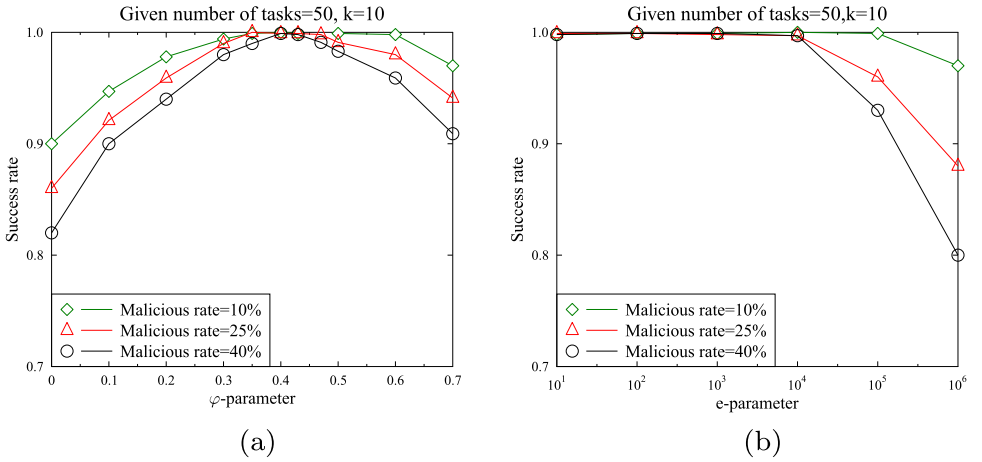

Fig. 7. Success rate impacted by φ and e parameters.

5.3.1. Impact of φ and e parameters

Parameter φ makes the ISAT approach more flexible and adaptable to different datasets. To study the impact of φ on success rate, we set dataset with 1000 Web services as the test sample. Meanwhile, we set $k = 10$, $e = 10^3$, $c = 10^3$, and given number of tasks = 50. We change φ value from 0.1 to 0.7 with an interval of 0.1. In the experiment, we employ three datasets with malicious rate 10%, 25% and 40%, respectively.

In Fig. 7(a), we observe that the success rate dramatically grows when the φ value increases from 0.1 to 0.4. This is because a higher φ value means that there may be more available services that are used in the selection process, which could be beneficial for an obvious increase on the success rate in malicious environments. However, the success rate rapidly reduces when the φ value is beyond 0.4 especially in the scenario where the number of malicious Web services is large since a higher φ value means that there may be less services that meet such a high interest value, which could result in decrease on the success rate. In addition, we also observe that the number of malicious Web services has obvious impact on the success rate. This result is expected because these unreliable or malicious Web services may directly cause failures in service-based composite systems. Therefore, we commonly adjust a feasible φ value to different datasets with the experiment-based method.

To investigate the impact of environment parameter e on the success rate, we introduce an environmental workload, which describes the workload at the time of invocation. In this experiment, we set $k = 10$, $\varphi = 0.4$, $c = 10^3$, and given number of tasks = 50. We vary e from 1 to 10^6. Fig. 7(b) shows the relation between e and the success rate with malicious rate 10%, 25% and 40%, respectively. The success rate dramatically decreases when e is beyond 10^4. This point is normally defined as a turning point whereas the workload beyond it could result in system shutdown or service crash. Therefore, we have reason to believe that we should avoid the workload to exceed the turning point in most cases.

5.3.2. Impact of c and k parameters

To study the impact of parameter c on success rate in an understandable way, we employ price as the only one c specific parameter to avoid coupling produced influences by multiple parameters. In this experiment, we set $k = 10$, $\varphi = 0.4$, $e = 10^4$, and given number of tasks = 50. Meanwhile, we vary c from 1 to 10^6. Fig. 8(a) shows the relation between c and the success rate under different numbers of malicious services. The success rate rises as the parameter c grows since the parameter c is aimed at obtaining coherent services with those original services customized by most users. Nevertheless, the success rate decreases when the parameter c value is beyond 10^3 on the same condition. This is because these services with a higher price c value may not be concerned by most users. This negative impact on success rate is more significant on the dataset with a higher malicious rate than with a lower malicious rate. Therefore, in practice, we normally calculate the average price of services for a certain dataset.

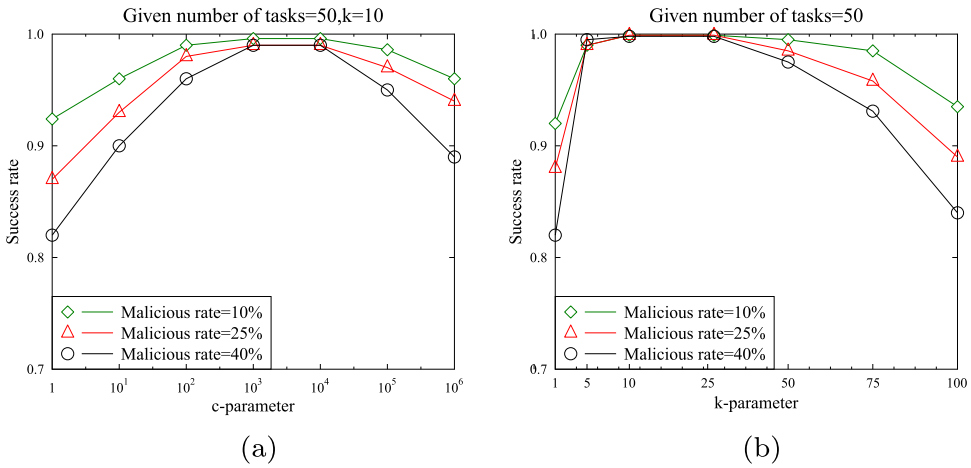

Fig. 8. Success rate impacted by c and k parameters.

To research the impact of top-k value on the success rate, we conduct the experiment in the same environment. We set $\varphi = 0.35$, $e = 10^3$ and $c = 10^3$. Fig. 8(b) shows the results of success rate with k value from 1 to 100. In this experiment, we observe that the success rate rapidly grows when the k value increases from 1 to 10. This is because a higher k value means that a greater number of alternative services are involved in selection, which could directly result in an increase on success rate. Then, the success rate maintains almost 1 when the k value rises from 10 to 25. This is expected because this range of k value basically covers on high-quality and high-usability services for each task. Finally, we also observe that the success rate reduces slowly when the k value continually increases beyond 30 because those services with high resource consumption or low usability are more likely to be chosen, which may directly result in the risk of reduction on success rate. Therefore, we usually adapt the reasonable range of k value for the sake of enhancing the success rate according to the characteristics of different datasets.

From all the experiments discussed above, we conclude that the approach ISAT is not only efficient, but also guarantees reliability in a malicious environment.

6. Discussion and related work

Many previous studies have presented various approaches of Web service selection for building reliable service-oriented softwares. We discuss only some of the notable work.

For efficient Web service selection, Zeng et al. [43] proposed a QoS-aware middleware for service selection and composition. They use an integer programming (IP) method to address the problem of global optimization in Web service selection. This method is not suitable for large-size problems because it may have a barrier in exponential time complexity especially when the service candidates grow in size. In [7], Alrifai and Risse employed MIP (Mixed Integer Programming) to obtain the optimal decomposition result of global QoS constraints into local constraints. Then, they select the best service for each given task based on the local constraints obtained. This method can significantly reduce time complexity, yet the decomposition algorithm could not ensure the local constraints to be suitable enough for local selection since different tasks have different QoS range values. Akbar et al. [2] mapped the QoS-aware service selection problem as multiple multidimensional multiple-choice knapsack problem (MMKP) and proposed a convex hull based algorithm for solving MMKP, called C-HEU. In their work, a flexible sorting method is initially introduced to select items with higher QoS performance and lower resource consumption for upgrading. The algorithm has quadratic time complexity and is especially fit for scenarios where the QoS value requested is not proportional to the resource consumption. Once the number of service candidates is very large and the resource consumption follows the monotone feasibility property, the algorithm could not obtain optimal solution. In addition, the above approaches typically considered an aggregated QoS metric, however the reliability of resulting service-oriented softwares may not satisfy user requirements.

For reliable Web service selection, a number of research efforts have been performed in the field of service selection related to reliability (*e.g.,* fault tolerance). In [47], Zheng and Lyu formulated the user requirements as local and global constraints and modeled the service selection for fault tolerance in service-oriented systems as an optimization problem. Then, they designed a heuristic algorithm to solve the optimization problem, called G-HEU. The success of G-HEU, including the approaches discussed above, depends on the correctness and precision of the supplied QoS data to a certain extent. If some QoS information provided is authentic, these algorithms can achieve an excellent performance. However, in the service-oriented environment, a malicious service provider may fail partially or fully in delivering the promised QoS at runtime [39]. Therefore, the approaches of Web service selection are required to distinguish what the authenticity of QoS is and what the maliciousness of QoS is *i.e.,* distinguish malicious services. Li et al. [23] proposed a trustworthy service

selection approach, called T-HEU. First, they designed a malicious (untrustworthy) service filter by a subjective threshold decision. Then, they built convex hulls for search space deduction. Finally, in the sub-optimization space, they used M-HEU [3] based heuristic algorithm to obtain the near-optimal solution. T-HEU has a better feature of scalability than other heuristic methods in terms of the polynomial time complexity, and is suitable for the situations where service candidates have stable QoS values and their QoS values do not dramatically fluctuate since the filter designed is based on the subjective threshold judgement, and a huge fluctuation in QoS values may result in the decrease on success rate. Moreover, the convex hull based heuristic algorithm for search space reduction could not obtain the desired near-optimal solution in the sub-optimization space. That is because, the deduction of search space requires more rigorous rules to restrain the services to be removed are not dominant than the convex hull based examples presented in their work. Once the dominant services are removed, the final solution could not be near-optimal.

For performance trade-off Web service selection, the research efforts primarily focus on how to achieve compromised performance of service selection to further satisfy the personalized users' requirements. Fletcher et al. [16] proposed a service selection method based on fuzzy logic, which considers users' personalized preferences on non-functional attributes and their trade-offs to select services. Zhang et al. [44] proposed an approach of service selection with user preferences via fuzzy weights, which can provide satisfactory services for users with varying cognitive levels. The above researches commonly find appropriate performance trade-offs of service selection among user preferences on each QoS attribute. Instead, the proposed service selection approach focuses on how to optimally select reliable services by the given precise and quantitative constraints excluding the user preferences.

Compared to these approaches, although we propose a service selection approach via two-phase decisions by considering multiple QoS attributes (generic QoS attributes and specific domain QoS attributes), we tackle the problem of service selection from a totally different perspective. First, we identify the generic and specific QoS attributes and construct the QoS model. Then, we make the first-phase decision to filter reliable services by employing the TOPSIS approach. Finally, we exploit the second-phase decision to select almost optimal services.

In addition, there are some complementary descriptions needed. Different from the convex hull based approach [14] which focuses on removing outliers that are outside any relevant facet and matching point sets refined, we employ the convex hull technique to identify dominant services to reduce the search space for Web services. Also different from the convex hull work in [23], which only illustrates some special cases for the deduction of search space, we reduce the search space based on the rigorous rules designed. Moreover, we typically consider the situation that the resource consumption follows the monotone feasibility property mentioned in [2] and build the corresponding rules to improve the performance.

7. Conclusion

This paper solved the problem of how to optimally select Web services for building reliable service-oriented softwares. Complementary to previous fault tolerance approaches, we proposed a decision-based approach ISAT to guarantee reliable Web services then select services for an optimal execution plan. The research contributions described in the paper are summarized as follows.

In the first-phase decision, we modeled the problem of identifying reliable service candidates as a MCDM problem. Then, we proposed the improved TOPSIS to address this problem. Compared with traditional decision approaches, our customized decision typically considered multiple decision parameters from different dimensions according to the characteristics of service-oriented systems. In the second-phase decision, we defined the problem of selecting services for an optimal execution plan as the specific 0–1 integer programming problem. Then, we proposed a convex hull based approach to efficiently solve the problem. Unlike traditional decision approaches, the convex hull based decision can reduce the search space of services to the minimum while ensuring the success rate and accuracy of the solution.

For the future work, we may consider several aspects to further improve the performance of the proposed ISAT including the first-phase decision and the second-phase decision model. For further improving the accuracy of the first-phase decision model, we will cluster more decision metrics from different dimensions of Web services into some representative ones using an improved k-means clustering method. For speeding up the second-phase decision model, we will try additional techniques, such as random walk and heuristic search, to achieve less computation time and more accurate solution. Moreover, for the experiments, we may combine MAE (Mean Absolute Error) or NDCG (Normalized Discounted Cumulative Gain) metric to measure the overall decision accuracy from different aspects. Our future work also includes how to detect and exclude malicious information with inaccurate non-functional descriptions. In addition, we will consider more functionality discussion including service consistency checking [18] and interface matching [19] when organizing services to replace the failed ones for further enriching our current research work. For example, we may employ interface matching including interface compatibility analysis for the first-phase decision to guide the proposed ISAT to select adequate service candidates. Then, it is necessary for the second-phase decision to check data consistency for better satisfying users' requirements. Furthermore, we will also try to extend the ISAT to support current emerging Web 2.0-enabled registries such as *Mashup* [48] and *ProgrammableWeb* [9].

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China (Grant No. 61428201) and the National Science Foundation-Career of USA (Grant No. 1622292).

References

[1] E. Al-Masri, Q.H. Mahmoud, Investigating web services on the world wide web, in: The 17th iInternational Conference on World Wide Web, 2010, pp. 795–804, doi:10.1145/1367497.1367605.
[2] M. Akbar, M. Rahman, M. Kaykobad, E.G. Manning, G.C. Shoja, Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls, Comput. Oper. Res. 33 (5) (2006) 1259–1273, doi:10.1016/j.cor.2004.09.016.
[3] M.M. Akbar, E. Manning, G. Shoja, S. Khan, Heuristic solutions for the multiple-choice multi-dimension knapsack problem, in: International Conference on Computational Science, 2001, pp. 659–668, doi:10.1007/3-540-45718-6_71.
[4] H. Al-Helal, R. Gamble, Introducing replaceability into web service composition, IEEE Trans. Serv. Comput. 7 (2) (2014) 198–209, doi:10.1109/TSC.2013.23.
[5] E. Al-Masri, Q.H. Mahmoud, QoS-based discovery and ranking of web services, in: 16th International Conference on Computer Communications and Networks, 2007, pp. 529–534, doi:10.1109/ICCCN.2007.4317873.
[6] M. Almulla, K. Almatori, H. Yahyaoui, A QoS-based fuzzy model for ranking real world web services, in: IEEE International Conference on Web Services, 2011, pp. 203–210, doi:10.1109/ICWS.2011.43.
[7] M. Alrifai, T. Risse, Combining global optimization with local selection for efficient QoS-aware service composition, in: ACM 18th International Conference on World Wide Web, 2009, pp. 881–890, doi:10.1145/1526709.1526828.
[8] D. Baski, S. Misra, Metrics suite for maintainability of extensible markup language web services, IET Softw. 5 (3) (2011) 320–341, doi:10.1049/iet-sen.2010.0089.
[9] S. Chen, Y. Fan, W. Tan, J. Zhang, B. Bai, Z. Gao, Time-aware collaborative poisson factorization for service recommendation, in: IEEE International Conference on Web Services, 2016, pp. 196–203, doi:10.1109/ICWS.2016.33.
[10] S.M. Chen, S.H. Cheng, T.C. Lan, Multicriteria decision making based on the TOPSISmethod and similarity measures between intuitionistic fuzzy values, Inf. Sci. 367 (2016) 279–295, doi:10.1016/j.ins.2016.05.044.
[11] M. Crasso, C. Mateos, Z. Alejandro, C. Marcelo, Easysoc: making web service outsourcing easier, Inf. Sci. 259 (2014) 452–473, doi:10.1016/j.ins.2010.01.013.
[12] G. Deconinck, V. De-Florio, O. Botti, Software-implemented fault-tolerance and separate recovery strategies enhance maintainability, IEEE Trans. Reliab. 51 (2) (2002) 158–165, doi:10.1109/TR.2002.1011520.
[13] K. Elgazzar, H.S. Hassanein, P. Martin, Daas: cloud-based mobile web service discovery, Pervasive Mob. Comput. 13 (2014) 67–84, doi:10.1016/j.pmcj.2013.10.015.
[14] J.F. Fan, Y.T. Zhao, D. Ai, Y.H. Liu, G. Wang, Y.T. Wang, Convex hull aided registration method (charm), IEEE Trans. Vis. Comput. Graph. 23 (9) (2017) 2042–2055, doi:10.1109/TVCG.2016.2602858.
[15] P. Felber, P. Narasimhan, Experiences, strategies, and challenges in building fault-tolerant CORBA systems, IEEE Trans. Comput. 53 (5) (2004) 497–511, doi:10.1109/TC.2004.1275293.
[16] K.K. Fletcher, X.Q. Liu, M.D. Tang, Elastic personalized non-functional attribute preference and trade-off based service selection, ACM Trans. Web 9 (1) (2015) 1–27, doi:10.1145/2697389.
[17] G. Friedrich, M.G. Fugini, E. Mussi, B. Pernici, G. Tagni, Exception handling for repair in service-based processes, IEEE Trans. Softw. Eng. 36 (2) (2010) 198–215, doi:10.1109/TSE.2010.8.
[18] H.H. Gao, Y.C. Duan, H.K. Miao, Y.Y. Yin, An approach to data consistency checking for the dynamic replacement of service process, IEEE Access 5 (2017) 1–12, doi:10.1109/ACCESS.2017.2715322.
[19] M. Garriga, A.D. Renzis, I. Lizarralde, A. Flores, C. Mateos, A. Cechich, A. Zunino, A structural-semantic web service selection approach to improve retrievability of web services, Inf. Syst. Front. 5 (2016) 1–26, doi:10.1007/s10796-016-9731-1.
[20] W.K. Haneveld, L. Stougie, M.H. Vlerk, An algorithm for the construction of convex hulls in simple integer recourse programming, Ann. Oper. Res. 64 (1) (1996) 67–81, doi:10.1007/BF02187641.
[21] A. Immonen, A. Pakkala, A survey of methods and approaches for reliable dynamic service compositions, Serv. Oriented Comput. Appl. 8 (2) (2014) 129–158, doi:10.1007/s11761-013-0153-3.
[22] R. Jhawar, V. Piuri, M. Santambrogio, Fault tolerance management in cloud computing: a system-level perspective, IEEE Syst. J. 7 (2) (2013) 288–297, doi:10.1109/JSYST.2012.2221934.
[23] J. Li, X.L. Zheng, S.T. Chen, W.W. Song, D.R. Chen, An efficient and reliable approach for quality-of-service-aware service composition, Inf. Sci. 269 (2014) 238–254, doi:10.1016/j.ins.2013.12.015.
[24] Z.N. Li, X.D. Yang, A reliability-oriented web service discovery scheme with cross-layer design in MANET, in: IEEE International Conference on Web Services, 2016, pp. 404–411, doi:10.1109/ICWS.2016.59.
[25] L. Liberti, C. Lavor, N. Maculan, A. Mucherino, Euclidean distance geometry and applications, SIAM Rev. 56 (1) (2014) 3–69, doi:10.1137/120875909.
[26] W. Lu, W.D. Wang, E. Bao, L.Q. Wang, W.W. Xing, Y. Chen, FAQS: fast web service composition algorithm based on QoS-aware sampling, IEICE Trans. Fundam. Electron.Commun. Comput. Sci. E99.A (4) (2016) 826–834, doi:10.1587/transfun.E99.A.826.
[27] S. Park, L. Liu, C. Pu, M. Srivatsa, J. Zhang, Resilient trust management for web service integration, in: IEEE International Conference on Web Services, 2005, pp. 506–513, doi:10.1109/ICWS.2005.99.
[28] K.L. Peng, C.Y. Huang, Reliability evaluation of service-oriented architecture systems considering fault-tolerance designs, J. Appl. Math. 14 (2014) 1–11, doi:10.1155/2014/160608.
[29] Z.U. Rahman, O.K. Hussain, F.K. Hussain, Time series QoS forecasting for management of cloud services, in: IEEE 9-th International Conference on Broadband and Wireless Computing, Communication and Applications, 2014, pp. 183–190, doi:10.1109/ICWS.2005.99.
[30] A.D. Renzisa, M. Garrigaa, A. Floresa, A. Cechicha, C. Mateosb, A. Zuninob, A domain independent readability metric for web service descriptions, Comput. Stand. Interfaces 50 (2016) 124–141, doi:10.1016/j.csi.2016.09.005.
[31] J. Serrano-Guerrero, J.A. Olivas, F.P. Romero, E. Herrera-Viedma, Sentiment analysis: a review and comparative analysis of web services, Inf. Sci. 311 (2015) 18–38, doi:10.1016/j.ins.2015.03.040.
[32] M. Smit, E. Stroulia, Simulating service-oriented systems: a survey and the services-aware simulation framework, IEEE Trans. Serv. Comput. 6 (4) (2013) 443–456, doi:10.1109/TSC.2012.15.
[33] K. Tamilarasi, M. Ramakrishnan, Design of an intelligent search engine-based UDDI for web service discovery, in: IEEE International Conference on Recent Trends In Information Technology, 2012, pp. 520–525, doi:10.1109/ICRTIT.2012.6206753.
[34] A. Vogel, B. Kerherve, G.V. Bochmann, J. Gecsei, Distributed multimedia and QoS: a survey, IEEE Multimed. 2 (2) (1995) 10–19, doi:10.1109/93.388195.
[35] D. Wang, L. Zou, D.Y. Zhao, Top-k queries on RDF graphs, Inf. Sci. 316 (2015) 201–217, doi:10.1016/j.ins.2015.04.032.
[36] F. Wang, K.Y. Xing, M.C. Zhou, A robust deadlock prevention control for automated manufacturing systems with unreliable resources, Inf. Sci. 345 (2016) 243–256, doi:10.1016/j.ins.2016.01.057.
[37] S.G. Wang, Z.B. Zheng, Z.P. Wu, M.R. Lyu, F.C. Yang, Reputation measurement and malicious feedback rating prevention in web service recommendation systems, IEEE Trans. Serv. Comput. 8 (5) (2015) 755–767, doi:10.1109/TSC.2014.2320262.

[38] W.D. Wang, L.Q. Wang, W. Lu, A resilient framework for fault handling in web service oriented systems, in: IEEE International Conference on Web Services, 2015, pp. 663–670, doi:10.1109/ICWS.2015.93.
[39] W.D. Wang, L.Q. Wang, W. Lu, An intelligent QoS identification for untrustworthy web services via two-phase neural networks, in: IEEE International Conference on Web Services, 2016, pp. 139–146, doi:10.1109/ICWS.2016.26.
[40] S.K. Yang, A condition-based failure-prediction and processing-scheme for preventive maintenance, IEEE Trans. Reliab. 52 (3) (2003) 373–383, doi:10.1109/TR.2003.816402.
[41] S.S. Yau, N. Ye, H.S. Sarjoughian, D.Z. Huang, M.A. Muqsith, Toward development of adaptive service-based software systems, IEEE Trans. Serv. Comput. 2 (3) (2009) 247–260, doi:10.1109/TSC.2009.17.
[42] T. Yu, Y. Zhang, K.J. Lin, Efficient algorithms for web services selection with end-to-end QoS constraints, ACM Trans. Web 1 (1) (2007) 1–25, doi:10.1145/1232722.1232728.
[43] L.Z. Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, H. Chang, QoS-aware middleware for web services composition, IEEE Trans. Softw. Eng. 30 (5) (2004) 311–327, doi:10.1109/TSE.2004.11.
[44] L.Y. Zhang, S.G. Wang, R.K. Wong, F.C. Yang, R.N. Chang, Cognitively adjusting imprecise user preferences for service selection, IEEE Trans. Netw. Serv. Manage. PP (99) (2017) 1–14, doi:10.1109/TNSM.2017.2731050.
[45] X.M. Zhang, X. Teng, H. Pham, Considering fault removal efficiency in software reliability assessment, IEEE Trans. Syst. Man Cybern.Part A 33 (1) (2003) 114–120, doi:10.1109/TSMCA.2003.812597.
[46] Z. Zheng, Y. Zhang, M.R. Lyu, Investigating qos of real-world web services, IEEE Trans. Serv. Comput. 7 (1) (2014) 32–39, doi:10.1109/TSC.2013.23.
[47] Z.B. Zheng, M.R. Lyu, Selecting an optimal fault tolerance strategy for reliable service-oriented systems with local and global constraints, IEEE Trans. Comput. 64 (1) (2015) 219–232, doi:10.1109/TC.2013.189.
[48] Y. Zhong, Y.S. Fan, K. Huang, W. Tan, J. Zhang, Time-aware service recommendation for mashup creation, IEEE Trans. Serv. Comput. 8 (3) (2015) 356–368, doi:10.1109/TSC.2014.2381496.