Hindawi
Complexity
Volume 2019, Article ID 9871971, 16 pages
https://doi.org/10.1155/2019/9871971

Research Article

Location-Aware Web Service Composition Based on the Mixture Rank of Web Services and Web Service Requests

Junwen Lu,[1] Guanfeng Liu ,[2] Keshou Wu,[1] and Wenjiang Qin[3]

[1]*Engineering Research Center for Software Testing and Evaluation of Fujian Province, Xiamen University of Technology, Xiamen, China*
[2]*Department of Computing, Macquarie University, Sydney, NSW, Australia*
[3]*Petro China Northwest Sales Company, China*

Correspondence should be addressed to Guanfeng Liu; guanfeng.liu@mq.edu.au

Received 24 January 2019; Accepted 20 March 2019; Published 7 April 2019

Guest Editor: Jianxin Li

Web service composition is widely used to extend the function of web services. Different users have different requirements of QoSs (Quality of Services) making them face many problems. The requirement of a special QoS may be a hard requirement or a soft requirement. The hard requirement refers to the QoS which must be satisfied to the user, and the soft one means that the requirement is flexible. This paper tries to solve the service composition problem when there are two kinds of requirements of QoSs. To satisfy various kinds of requirement of the QoS, we propose a composition method based on our proposed framework. We give an analysis from composition models of services and from related QoE (Quality of Experience) of web services. Then, we rank the service candidates and the service requests together. Based on the ranking, a heuristics is proposed for service selection and composition-GLLB (global largest number of service requests first, local best fit service candidate first), which uses "lost value" in the scheduling to denote the QoE. Comparisons are used to evaluate our method. Comparisons show that GLLB reduces the value of *NUR* (Number of Unfinished service Requests), *FV* (Failure Value), and *AFV* (Average Failure Value).

1. Introduction

In recent years, high-performance hardware and software resources make the Cloud widely used by companies and departments. Cloud computing is based on Visualization and Service-Oriented Architecture (SOA). The goal of Cloud computing is to optimize the usage of physical and software resources, improve flexibility, and automate management [1]. When web services come into Cloud, they enhance services by service composition. The QoSs of the web service decides whether the Cloud can satisfy the requirement of the user. Typically, QoS requirements include execution time, scalability, high availability, trust, security, and so on [2].

A successful web service in service-oriented Cloud computing, not only tries to provide functionality as request, but also ensures to satisfy the requirement of QoSs [3]. Different users have diverse requirements to various QoSs [4]. For example, when users download files from the Internet, they always prefer to a web service with higher bandwidth; but when users transform money between banks, security is the most important QoS.

A web service only provides a single functionality for user. Web service composition enhances the ability of the Cloud, which combines multiple web services together to form a new functionality. At the same time, it also brings problems for the scheduling of web services [5]. Quality of Experience (QoE) is a measure of the level of customer satisfaction of a service provider [6]. While QoS is to illustrate the quality of services, which contains several attributes of the services. When users have many service composition requests, the selection order of web services influences the scheduling result of others, which have diverse value in QoE.

A. Jula et al. summarize nine targets for the web service selection [7], including (1) user requirement satisfaction; (2) qualitative to quantitative transformation of QoS parameters; (3) algorithm improvements; (4) introducing data storage and indexing structures; (5) self-adaptability, automaticity, increasing reliability, and accuracy, and quality assurance; (6)

proposing an improvised QoS mathematical model; (7) revenue maximization; (8) optimization of the service discovery process; (9) proposing new frameworks and structures. Most of time, those nine targets are influenced by each other, such that (3) algorithm improvements always influence (5) self-adaptability. In this paper, we try to consider multiple aspects, especially for (1), (2), (3), and (8).

When some requirements of QoS attributes are flexible (soft), the service composition problem becomes more difficult [8]. Because the flexible requirement of QoS attributes makes the selection of web service more widely, and it is difficult to decide which web service is the right selection. For example, if we assign a web service request with a web service which has a higher value in QoS attribute than the requirement, it may make others web service requests which need higher values in QoS attribute cannot be satisfied. If we assign a web service request with a web service which just satisfies the lowest requirement, the system may have low performance for the whole system, and the service may have lower value in QoE, especially when there are enough web services. The main problem of this paper is to schedule the web services when (1) the web service has different QoS attributes; (2) the service composition request has various requirements to diverse QoS attributes (hard or soft); (3) there are different functions for QoE to QoS for various kinds of requirements.

The main contributions of the paper are as follows: (1) we give a framework of web service composition; (2) we discuss web service composition methods from the QoE; (3) a new ranking of web services and requests is proposed; (4) a heuristic is proposed for web service selection and composition.

The rest of the paper is organized as follows. In the next section, we give an overview of web services from various aspects, such as the attributes, the preference of users, and web service composition methods. Section 3 describes the framework of web service composition. Section 4 discusses the service composition from the QoE. Section 5 gives a new ranking method for web services and requests. At the same time, the section also proposes a web service selection and a composition heuristic. Section 6 explains the simulation environment and the simulation results. Section 7 presents the conclusion and the future work.

2. Related Work

The difficulty of web service composition comes from many aspects. Those reasons are as follows:

(1) There are many kinds of QoS attributes about the web services [8, 9]. Those QoS attributes include execution time, reliability, availability, reputation, price, trust degree, and so on. Even a QoS attribute consists of multiple subattributes [2].

(2) There are many structures for the service composition request; the basis structures include sequence structure, parallel structure, loop structure, and case structure. And for a real service composition request, it is always a mixture of those structures that makes the service composition problem more difficult.

(3) The value of QoS attribute also has different kinds and it is difficult to get the accuracy value. The value may belong to Boolean, numerical, or even a scope. Some QoS attributes belong to objective and others belong to subjective. For the objective value, such as reputation, it is difficult to get the accurate value of the related QoS attributes.

(4) Some QoS attributes influence each other [10]. Service-dependent QoSs can be partial and full dependencies. One example is the execution time which always influences the reputation.

(5) Different users have diverse preferences to QoS attributes [4, 11–13]. Because of various targets of different users, users have diverse preferences to the QoS. In other words, even for the same QoS attributes, different users have unlike QoEs [14].

Because of the above-mentioned reasons, service composition is difficult. Researchers also have proposed some methods from different aspects.

Some researchers have given some service composition methods through recommendation. C. Xi et al. [15] use a novel collaborative filtering-based web service recommender system to help users to select the web service. The system takes account of the location information and QoS values and then clusters users and services. A recommendation is suggested based on the Cluster result. Q. Yu [16] proposes CloudRec to find out different community of users and services from large scale data. The service community is based on the inherent Cloud-related features, and it can help to estimate the QoS of unknown services. W. Denghui et al. [17] divide the QoS into two categories: global and local. Different methods are used to the two categories for a reliable web service composition recommendation approach. Global QoS dimensions are used to calculate the global QoS value of service composition. Then the credibility of service composition is computed with the distance of the execution result and the advertised QoS value. The lowest QoS value in all service units is selected in the local QoS dimension. The credibility of local QoS dimensions is got from the credible user experience. Web service evaluation result is the composition of the weight of user preference with the credible QoS information. The service with the highest score is chosen in the service selection.

Considering the preference of different users, researchers also give some new composition methods. To maximize (or at least ensure an acceptable) QoE, while ensuring fairness among users, H. Tobias et al. [4] propose a QoE fairness index F to solve this problem. S. Zhang et al. [18] try to help the user to find the top k composition services to satisfy the composition request, which has diverse preferences to different kinds of QoS. The preference-aware service dominance is used to denote the preference. A multi-index based algorithm is used to calculate the local service selection and top-k composite services are selected under a dominant number-aware service ranking in global optimization. According to

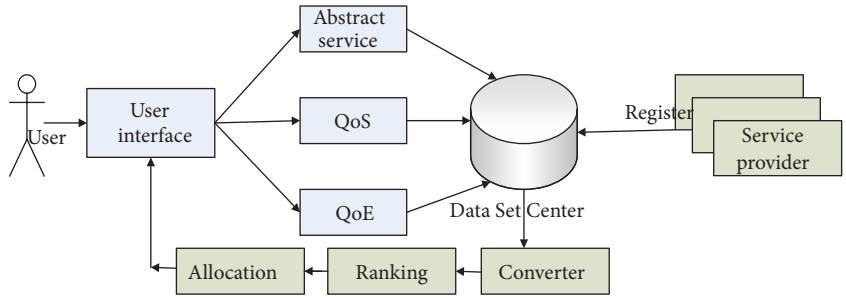


Figure 1: System overview of service composition.

the QoS preference of a user, H. Wang et al. [19] propose to integrate both the quantitative and qualitative preference to calculate the user similarity. They seek similar users by using user similarity service recommendation based on the idea of collaborative filtering.

Evolutionary algorithms are widely used in the web service composition. Y. Yu et al. [20] try to solve the service composition problem under distributed environment. They use a network coordinate system to estimate the time on the communication link between service providers and end users. And they propose a new GP-based (Genetic Programming) approach which considers multiple QoS constraints simultaneously. Z. Ding et al. [21] try to select and compose web services via a genetic algorithm (GA) and they combine those QoSs as a fitness function. They also give a transaction and QoS-aware selection approach. C. Cremene et al. [22] perform an analysis of several state-of-the-art multiobjective evolutionary algorithms. Those methods include Nondominated Sorting Genetic Algorithm II (NSGA-II), Strength Pareto Evolutionary Algorithm 2 (SPEA2), Multiobjective Multistate Genetic Algorithm (MOMS-GA), Differential Evolution (DE), multiobjective DE version called Generalized Differential Evolution (GDE3), and so on. Simulation results show that GDE3 algorithm yields the best performances on this problem.

Even the same value of QoS, various users have diverse QoEs. The reason is that different users have different targets. QoE also has been used in the service composition. Most of research focuses on the relation between the QoS and QoE. B. Upadhyaya et al. [6] provide an approach to extract a user's perception of the QoSs from user reviews on services (QoEs) and use such information to compose services. According to the QoEs of different users, they select a particular service from a pool of services and recommend the best service execution path in a composited service.

Some researchers also pay attention to the fact that some requirements of QoSs are flexible [8, 14]. In other words, if the value of the QoS attribute cannot satisfy the requirement, it just loses something, such as QoE and reputation. C. Lin et al. [8] proposed a QoS-based service selection (RQSS) algorithm to help users to find feasible web services according to the functionalities and the QoS of users' requirements. If no web service could exactly satisfy users' requirements, the RQSS recommends prospective service candidates to users by relaxing QoS constraints.

Other methods, such as ant colony [23] algorithm, liner approaches [24], and social network management [25], are also used in the service composition. Q. Wu et al. [23] model the problem of service composition problem as a constrained directed acyclic graph. They use the ant colony optimization algorithm to seek a near-to-optimal solution. S. Bharathan et al. [24] address the service composition in a multicloud environment. They propose an Integer Linear programming model where violations in global constraints are permitted but with an associated penalty. W. Chen et al. [25] use linked social service-specific principles based on linked data principles to publish services on the open web as linked social services. Based on complex network theories, they construct the global social service network following linked social service-specific principles. They used the global social service network to provide linked social services as a service.

Different from the prior work, we consider the environment when the requirements are a mixture of hard requirements and soft (relaxable) requirements. We try to discuss the service composition from the relation of QoS and QoEs, different from most of past work, which often composites services from the QoSs of the service. This also makes the task have diverse performance in QoE. S. Chakhar [26] just takes account of "hard" nonfunctional requirement in the scheduling after considering the function requirement. Different to [26], we try to consider the hard requirement and flexible (soft in [26]) requirement of functional requirement and nonfunctional requirement at the beginning. Most important of all, we consider the QoE of the service providers and service requesters together. We try to discuss the service composition from the QoEs of users, different from most of past work, which often composites services from the QoSs of the service.

3. SLA Service Composition Architecture

Our test data and the proposed method are implemented based on the framework shown in Figure 1. Users use "user interface" to submit the requirement of the abstract services, the requirement of QoSs, and the related QoE functions. The abstract services indicate which kind of services that users need. The requirement of the QoS illustrates the requirement of the value of QoS. The QoE function gives the relation between the QoS attribute and the QoE. In our paper, those requirements include hard requirement and soft requirement.

Service providers register service information to the system through UDDIe [27]. The information includes the identification of abstract services, the QoS of related services, the number of related services, and so on. The Data Set Center keeps and checks the information from the user and the service provider. The service provider registers service information (service name, QoSs) to the Data Set Center, and user provides the service requirement (service name, QoSs; QoEs) to Data Set Center through user interface. According to the QoS and the QoE function, Converter converts the QoS attribute value into QoE value. Different kinds of QoS requirement have different QoE function. Different to most of research, this paper ranks the service request and the service provider together which helps us to make decision in the Allocation (Section 4). Allocation schedules the web service providers to the web service requests (Section 5.3). The three modules (Converter, Ranking, and Allocation) work together to provide SLA (Service-Level Agreement) service composition for provider. A service-level agreement is a commitment between a service providers and users. SLA also supports the hard requirement and soft requirement. It makes various users may give diverse QoE value for the same QoS according to the SLA.

4. Web Service Ranking Based on Hard/Soft Requirement of Different Attributes

As discussed in Section 2, the web service has many attributes and even every attribute has many subattributes. The ranking problem is defined as multiple criteria decision making (MCDM). There are three fundamental approaches to solving MCDM problems: Analytic Hierarchy Process (AHP), Multiple Attribute Utility Theory (MAUT), and outranking. Most of methods are based on the three methods.

4.1. QoS Model. In the paper, we consider five QoSs and they are execution time, reliability, availability, reputation, and price [4, 8, 14, 17].

(i) Execution time: the execution time of the service request *sr* is the time from a service *s* has allocated to *sr* to the time that the *sr* has been finished.

(ii) Reliability: the reliability of the service *s* is the probability that a request is correctly responded as the request of the user in the expected time.

(iii) Availability: the availability of the service *s* is the probability that a service request exists or is ready to use within the expected time.

(iv) Reputation: reputation is used to reduce the impact of malicious attacks. The value of service reputation gets from customer's subjective scores and objective judgment to the credibility of QoS advertisement information (Bayesian Learning Theory) [4].

(v) Price: the cost is involved in requesting and using the service (Unit: USD/10 min).

More kinds of QoS, such as trust degree, operability, and delay, also can be added to our framework as the method introduced in the following.

There are M abstract services (formula (1)) and every abstract service has different numbers of candidate services. The abstract service s_m ($1 \le m \le M$) has ns_m candidate services (formula (2)). SC_m is the relative QoSs of different candidate services of the abstract service s_m. $sv(m, svid)$ denotes the $svid$th service candidate of the abstract service s_m (formula (3), $1 \le m \le M$). $QoS(m, svid)$ is the QoS set of the $svid$th service candidate of the abstract service s_m (formula (4), $1 \le svid \le ns_m$). $Et(m, svid)$, $Rel(m, svid)$, $Av(m, svid)$, $Rep(m, svid)$, and $Pr(m, svid)$ are the values of QoSs for the execution time, reliability, availability, reputation, and price, respectively.

$$S = \{s_1, s_2 \ldots, s_m, \ldots s_M\} \tag{1}$$

$$NS_m = \{ns_1, ns_2 \ldots, ns_m, \ldots ns_M\} \tag{2}$$

$$SC_m = \{sv(m,1), \ldots sv(m, svid), \ldots, sv(m, ns_m)\} \tag{3}$$

$$QoS(m, svid) = \{Et(m, svid), Rel(m, svid), Av(m, svid), Rep(m, svid), Pr(m, svid)\} \tag{4}$$

There are N service composition processes (requests) from the users (formula (5), $1 \le n \le N$). Formula (6) indicates which kind of abstract service the nth service composition process needs. If $sa(n, rid)$ equals 1 ($1 \le rid \le M$), it means that the nth service composition process needs the mth abstract service. At the same time, a vector of QoS requirements is listed in formula (7). $RQoS(n, rid)$ is the QoS requirement of nth service composition process to mth abstract service. $Ret(n, rid)$, $Rrel(n, rid)$, $Rav(n, rid)$, $Rrep(n, rid)$, and $Rpr(n, rid)$ are the requirement of the QoS of the execution time, reliability, availability, reputation, and price, respectively. Formula (8) computes the total number of candidate services to the abstract service s_m. M is the kinds of abstract services.

$$SP = \{sp_1, sp_2 \ldots, sp_n, \ldots sp_N\} \tag{5}$$

$$SA_n = \{sa(n,1), sa(n,2) \ldots, sa(n, rid), \ldots sa(n, M)\} \tag{6}$$

$$RQoS(n, rid) = \{Ret(n, rid), Rrel(n, rid), Rav(n, rid), Rrep(n, rid), Rpr(n, rid)\} \tag{7}$$

$$sum_m = \sum_{rid=1}^{M} sa(n, rid) \tag{8}$$

4.2. Different QoE to QoS. The value of QoE is a function to the related QoS of the candidate service. Figure 2 shows that some typical functions in the true system, X axis indicates the value of QoSs, and Y axis indicates the benefit for the user to the special QoS (QoE). Figures 2(a)–2(d) show the relations between the QoS and the QoE whose QoSs belong to benefit QoS (positive QoS) that higher value always brings better result. Figures 2(e)–2(h) show the QoS value that belongs to cost QoS (negative QoS) that higher value always brings negative effort to the user.

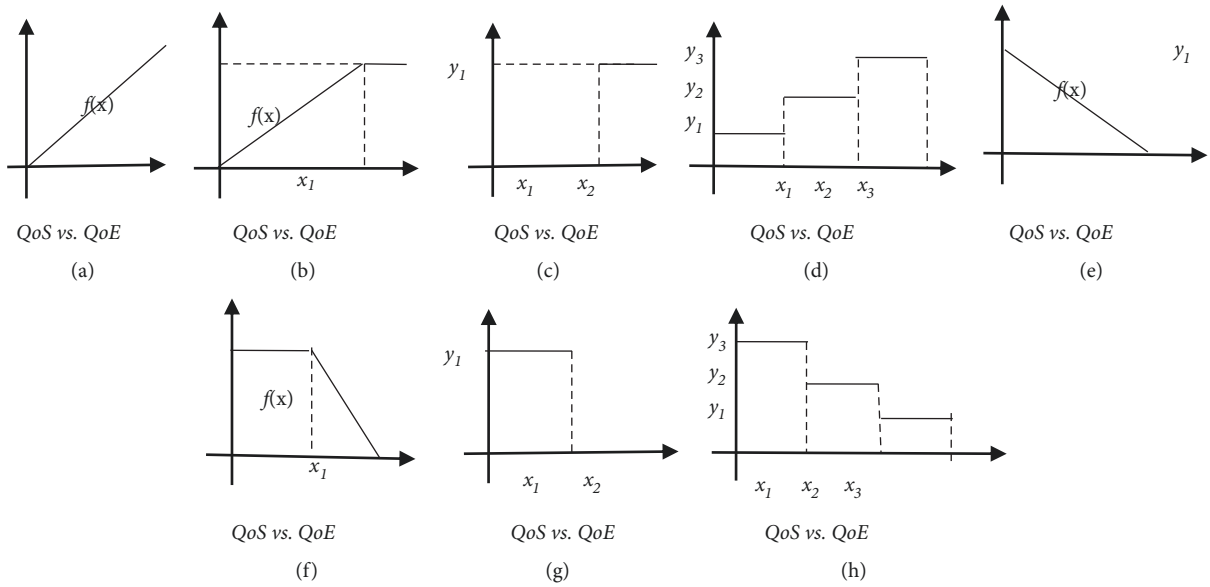


Figure 2: QoS versus QoE.

The QoE function $QoEv(m, svid, n, rid)$ of the QoS is

$$QoEv(m, svid, n, rid) = QoEf(QoS(m, svid), RQoS(n, rid)) \quad (9)$$

$QoS(m, svid)$ is the QoS set of the $svid$th service candidate of the abstract service s_m and $RQoS(n, rid)$ is the QoS requirement of nth service composition process to mth abstract service.

The value of QoE also can be categorized into four kinds: Boolean, numerical, unordered set, and range type. According to [28], same as the value of QoS, all of them can be transformed into numerical. In this paper, the hard requirement can be taken as the Figure 2(c) or Figure 2(g), and the soft requirement can be taken as Figure 2(b) or Figure 2(f).

5. QoS Normalization for QoSs of the Service Candidates and the Requirement of QoSs of the Service Requests

In this section, first of all, we normalize the QoS of service and the request to QoS of the service request, then we rank the service and service request; at last, a heuristic is proposed for service selection and composition.

5.1. QoS Normalization of the Service and the Service Request. The hypotheses in the paper are that, for the special abstract service s_m, the total number of the service candidates is ns_m (formula (2)), those service candidates are listed in the set SC_m (formula (3)), and the QoS of those service candidates is listed in the set $QoS(m, svid)$ (formula (4)); to the special service process sp_n, the request to different abstract services is listed in the set SA_n (formula (6)), and the requirement of the QoS of the abstract service is listed in the set $RQoS(n, rid)$ (formula (7)). Then the problem is how to assign services candidates to the service request under different kinds of requirement (hard and flexible) of the QoS. Past work mainly uses aggregation functions to schedule those services [4, 8, 9, 15]. A service composition request has many service requests to different abstract services. A service request refers to a detailed request to an abstract service. Before using the aggregation function, normalization of QoS is executed to make the values of different QoSs have the same scope (between 0 and 1).

Most of work for the normalization of the QoS is to the service, such as [8], not to the service request. In the paper, we consider two aspects simultaneously: QoS of service and the requirement of QoS of the service request. Such as execution time, we consider $Et(m, svid)$ $(1 \le svid \le ns_m)$ and $Ret(n, m)$ $(1 \le n \le N, 1 \le m \le M)$ as a new set ET (formula (10)). $Et(m, svid)$ and $Ret(n, m)$ are the requirement of execution time from the service requesters and the execution time from service providers.

$$ET = \{Et(m, svid) \mid 1 \le svid \le ns_m\} \cup \{Ret(n, m) \mid 1 \le m \le M\} \quad (10)$$

$Rel(n, m)$, $Rav(n, m)$, $Rrep(n, m)$, and $Rpr(n, m)$ mean the provided reliability, availability, reputation, and price from the services providers, respectively. $Rrel(n, m)$, $Rav(n, m)$, $Rrep(n, m)$, and $Rpr(n, m)$ mean the requirement of the QoS to reliability, availability, reputation, and price from service requesters, respectively. We make similar considerations for the other QoSs:

$$REL = \{Rel(m, svid) \mid 1 \le svid \le ns_m\} \cup \{Rrel(n, m) \mid 1 \le m \le M\}$$

$$\begin{aligned} AV &= \{Av(m, svid) \mid 1 \le svid \le ns_m\} \\ &\cup \{Rav(n, m) \mid 1 \le m \le M\} \\ REP &= \{Rep(m, svid) \mid 1 \le svid \le ns_m\} \\ &\cup \{Rrep(n, m) \mid 1 \le m \le M\} \\ PR &= \{Pr(m, svid) \mid 1 \le svid \le ns_m\} \\ &\cup \{Rpr(n, m) \mid 1 \le m \le M\} \end{aligned} \tag{11}$$

To make different values of QoSs of the service candidate and QoS requirements of the service request in the same range, Gaussian method [29, 30] is used to normalize them for feature extraction. It cannot only map those values to the interval [0, 1], but also better avoid the influence of abnormal values (such as high value or low value) compared with other normalization methods. The detail can be described as follows:

Suppose et_i^{Et} is an element of ET (formula (10)), we normalize it to no_i^{Et}:

$$no_i^{Et} = 0.5 + \frac{el_i^{Et} - \overline{ET}}{2 \times 3\sigma_{ET}} \tag{12}$$

where no_i^{Et} is normalized value of et_i^{Et}, $\overline{ET}$ is the average value of ET, and σ_{ET} is the standard deviation of ET. Prior work [30, 31] shows that 99% data can be mapped into [0, 1]. If the value is less than 0, we set it to 0; if the value is more than 1, we set it to 1.

Suppose no_i^{Rel}, no_i^{Av}, no_i^{Rep}, no_i^{Pr} are one element of REL, AV, REP, and PR, respectively. $\overline{REL}$, $\overline{AV}$, $\overline{REP}$, and $\overline{PR}$ are the average value of REL, AV, REP, and PR, respectively. σ_{REL}, σ_{AV}, σ_{REP}, and σ_{PR} are the standard deviation of REL, AV, REP, and PR, respectively. Similar to the method about execution time, we normalize the four elements as follows:

$$\begin{aligned} no_i^{Rel} &= 0.5 + \frac{el_i^{Rel} - \overline{REL}}{2 \times 3\sigma_{REL}} \\ no_i^{Av} &= 0.5 + \frac{el_i^{Av} - \overline{AV}}{2 \times 3\sigma_{AV}} \\ no_i^{Rep} &= 0.5 + \frac{el_i^{Rep} - \overline{REP}}{2 \times 3\sigma_{REP}} \\ no_i^{Pr} &= 0.5 + \frac{el_i^{Pr} - \overline{PR}}{2 \times 3\sigma_{PR}} \end{aligned} \tag{13}$$

If the value is not in the range [0, 1], we also use the same method to deal with the data to no_i^{ET}.

According to whether a larger value of QoS attribute means a better service, the QoS attributes can be categorized into two kinds: cost (negative) QoS or benefit (positive) QoS. Reliability, availability, and reputation belong to benefit QoS, execution time, and price belongs to cost QoS. For the cost QoSs, we transform them as follows so that make larger value of the related QoSs also means a better service:

$$\begin{aligned} not_i^{Et} &== 1 - no_i^{Et} = 1 - \left(0.5 + \frac{el_i^{Et} - \overline{ET}}{2 \times 3\sigma_{ET}}\right) \\ &= \frac{\overline{ET} - el_i^{Et}}{2 \times 3\sigma_{ET}} + 0.5 \\ not_i^{Pr} &= 1 - no_i^{Pr} = 1 - \left(0.5 + \frac{el_i^{Pr} - \overline{PR}}{2 \times 3\sigma_{PR}}\right) \\ &= \frac{\overline{PR} - el_i^{Pr}}{2 \times 3\sigma_{PR}} + 0.5 \end{aligned} \tag{14}$$

The above two formulas also have a scope of [0, 1].

5.2. Service Selection and Composition. For service composition, we give different weights to the five QoS attributes according to the condition of the whole system. Our model can extend to the services with more than five QoS attributes, but here we consider those important attributes only. The final rank of the service (or service request) S_m:

$$\begin{aligned} Sco_i^m = {} & W_{Et}^m \times not_i^{Et} + W_{Rel}^m \times no_i^{Rel} + W_{Av}^m \times no_i^{Av} \\ & + W_{Rep}^m \times no_i^{Rep} + W_{Pr}^m \times not_i^{Pr} \end{aligned} \tag{15}$$

i are the unique identifier of services (including service request and service provider). m is the unique identifier of abstract services. W_{Et}^m, $W_{Re;}^m$, W_{Av}^m, W_{Rep}^m, and W_{Pr}^m are the different weights of the five QoS attributes: execution time, reliability, availability, reputation, and price. Various researchers use diverse methods to set the weight. Those methods are classified into two kinds: weights decided by the user and those decided by the system. Our method belongs to the latter. Those parameters are listed as

$$\begin{aligned} tot = {} & \sum_{i=1}^{sum_m} Num_{i,m}^{Et} + \sum_{i=1}^{sum_m} Num_{i,m}^{Rel} + \sum_{i=1}^{sum_m} Num_{i,m}^{Av} \\ & + \sum_{i=1}^{sum_m} Num_{i,m}^{Rep} + \sum_{i=1}^{sum_m} Num_{i,m}^{Pr} \\ W_{Et}^m &= \frac{\sum_{n=1}^{sum_m} Num_{i,m}^{Et}}{tot} \\ W_{Rel}^m &= \frac{\sum_{i=1}^{sum_m} Num_{i,m}^{Rel}}{tot} \\ W_{Av}^m &= \frac{\sum_{i=1}^{sum_m} Num_{i,m}^{Av}}{tot} \\ W_{Rep}^m &= \frac{\sum_{i=1}^{sum_m} Num_{i,m}^{Rep}}{tot} \\ W_{Rp}^m &= \frac{\sum_{i=1}^{sum_m} Num_{i,m}^{Pr}}{tot} \end{aligned} \tag{16}$$

1. Calculate the number of all the service candidates (Sum_m);
2. $[M_{ID}, M_{value}] = sort(\{sum_m \mid 1 \le m \le M\})$;
   //sum_m is number of service candidates, sort sum_m in descending order;
   // M_{ID} is a set of the abstract service ID
   //M_{value} is a set of the number of the service requests to different abstract services.
3. For m=1:M
   3.1 $[mapc - r, unr]$ = schhard(abstract service s_m);
   //for the abstract service s_m, find a schedule method;
   //this step supposes that every requirement to the QoS is a hard requirement;
   //*mapc-r* is a map between the service candidate of the abstract service s_m and service request to the abstract service s_m;
   //*unr* is a service request set that cannot be satisfied;
   3.2 Delete the service process that includes the abstract service s_m and the request to the abstract service s_m cannot be satisfied;
   EndFor
4. For m=1:M
   4.1 $[mapc - r, unr]$ = schsoft(abstract service s_m)
   //this step considers the requirement to the QoS that is relaxable;
   //*mapc-r* is a map between the service candidate of the abstract service s_m and service request to the abstract service s_m;
   //*unr* is a service request set that cannot be satisfied;
   4.2 Delete the service process that includes the abstract service s_m and the request to the abstract service s_m that cannot be satisfied;
5. Endfor

Algorithm 1

where $\sum_{i=1}^{sum_m} Num_{i,m}^{Et}$ is the number of services which cannot satisfy the service request from the view of the execution time; sum_m (see formula (8)) is the total number of the request to the abstract service s_m.

$\sum_{i=1}^{sum_m} Num_{i,m}^{Rel}$, $\sum_{i=1}^{sum_m} Num_{i,m}^{Av}$, $\sum_{i=1}^{sum_m} Num_{i,m}^{Rep}$, and $\sum_{i=1}^{sum_m} Num_{i,m}^{Pr}$ are the numbers of services which cannot satisfy the service requests from the view of reliability, availability, reputation, and price, respectively.

5.3. Service Selection. There are two steps for the service selection: (1) the first step decides which abstract service will be allocated first; (2) the second step decides assign which service candidate to the service request of the selected abstract services.

Algorithm 1 gives the first step. First of all, we calculate the number of service requests to diverse abstract services (Line 1). Then, we sort the value in descending order. First of all, we schedule the abstract service which has the maximum value in the number of the service requests to different abstract service. The reason is if there are some service requests that cannot be satisfied, we can delete those service processes (Step 3.2, in Algorithm 1); those service processes maybe also include other service requests to other abstract services.

First of all, we suppose every requirement of the QoS attribute is a hard requirement; "*schhard*(abstract service s_m)" is used to find the service selection result (Steps 3.1 and 3.2, in Algorithm 1). Algorithm 2 gives the details. If there are service requests which cannot get the right services, soft requirement is considered and "*schsoft*(abstract service s_m)" is used to find the service selection result (Steps 4.1 and 4.2, in Algorithm 1). Algorithm 3 provides the details.

After we sort the SCO (line 3, in Algorithm 2) in descending order, we can consider the problem as two problems: selecting which service composition request and selecting which service candidate to the selected service request.

(1) The first problem decides which service request that we select first. "Largest First" is used to select the first service composition request. It means we always select the service request which has the largest value of the aggregation function. We select the "Largest First" because if the service request has a larger value in aggregation function, then it always has more difficult to find a service candidate that satisfies the requirement of the QoS attributes.

(2) The second problem decides which service candidate that we first try to check for the selected service request. "Local" means the service candidate whose position is near to the service request according to the value of aggregation function. In other words, there is no much difference in the value of aggregation function between those service candidates and the service requests. In the system, we give "Local" in this way: it has two directions, left or right. For the left, from the position of the service request, visiting left one by one, after it finds some service candidates and stops until it finds a service request or to the end of left again; for the right, from the position of the service request (Global view decides the service request), visiting right one by one, after it finds some service candidates or to the end of the right and stops until it finds a service request again. For the local view, we can check the best service (Local Best First Check) candidate. The service candidate who has a higher value of aggregation

```
//GLLB
1. Calculate the weights of different QoSs according to formulas in Sections 5.1 and 5.2;
2. Calculate the aggregation value for the service candidate and service request of the abstract
   service s_m;
3. SCO = {Sco_i^m | 1 ≤ i ≤ (ns_m + sum_m)};
4. [tp, ID, Val = sort(SCO)];
      // sort SCO in descending order;
      //tp can be a service candidate (="candidate") or a service request (="request");
      //ID records the ID for the service candidate or the service request;
      //Val records the value of aggregation function for the service candidate or the service
request;
5. Select the first service composition request sr
6. Get the left position lp and the right position rp;
7. minmore=+∞, selectr=-1;
8. For pos = lp : rp
        If the service candidate Sco_pos^m can satisfy sr for all the QoS
            minmoretemp = mvalue(Sco_pos^m , sr);
          If minmoretemp< minmore
            minmore=minmoretemp;
            selectr=Sco_pos^m;
          Endif
        EndIf
   Endfor
9. If selectr=-1
        Assign selectr to sr;
   EndIf
   //minmore records the total value of the QoS of the web service that which is more than the
requirement of the service request under the condition that all of requirements of the service
composition request have been satisfied.
10. pos = rp + 1;
        //search from the position of the end of the right position;
11. While (pos < (ns_m + sum_m))
            11.1 If Sco_pos^m can satisfy the requirement of QoS when we suppose every
            requirement is hard
                    selectr=Sco_pos^m;
                    Assign selectr to sr;
                    Break;
                EndIf
   EndWhile
```

Algorithm 2: Schhard (abstract service s_m).

value does not always mean it can satisfy the service request with lower aggregation value. Some service requests may have the hard requirement of the QoS. Even some service candidates which have a lower value of aggregation function also can satisfy the service request which has a higher value of aggregation function by relaxing some requirements of soft QoSs. If a service request cannot get a service candidate in the left and the right, it will search all service candidates.

In summary, our method is GLLB (global largest number of service request first, local "best fit" service candidate first). For the service composition request, the one which has the largest aggregation function value will be selected first; the service candidate, the one which is the "best fit", will be selected. "Best fit" has two aspects of meaning.

When we suppose every requirement of QoS attribute is hard, we select the service candidate that would have the lowest lost value (See formulas (18)~(22)). We suppose there are a web service candidate *wsc* and a web service request *wsr*, and QoSs of wsc and QoS requirement of *wsr* have been normalized. $hard_{wsr}$ indicates the requirement of the related QoS that is relaxable. If ETh_{wsr} equals 0, it is a hard requirement; otherwise (being equal to 1), it is a flexible requirement, similar to others QoS attributes. If there are many service candidates that can satisfy the requirement when we suppose they are hard requirements, we will select the service candidate which has the smallest value in *mvalue*(formula (20)), which has the smallest average lost value in the QoS. *moren* is the number of QoSs which is better than the requirement. Formula (21) calculates the lost value of execution time, same to all cost QoS attributes; formula (22) calculates the lost value of reliability, same to others benefit QoS attributes. If we cannot get a service candidate, we will go to the next step.

$$wsc = \{ET_{wsc}, Rel_{wsc}, Av_{wsc}, Rep_{wsc}, Pr_{wsc}\} \quad (17)$$

```
// GLLB
1. Calculate the weight of different QoS according to formulas in Sections 5.1 and 5.2;
2. Calculate the aggregation value for the service candidate and service request of the abstract
   service s_m;
3. SCO = {Sco_i^m | 1 ≤ (ns_m + sum_m)};
4. [tp, ID, Val] = sort(SCO);
      //sort SCO in descending order;
      //tp can be a service candidate (="candidate") or a service request (="request");
      //ID indicates the one for the service candidate or the service request;
      //Val indicates the value of aggregation function for the service candidate or the service
request;
5. Select service composition request sr that cannot find a service candidate in Algorithm 2
   (Algorithm 2)
6. Get the left position lp and the right position rp;
7. minlost=+∞, selectr=-1;
8. For pos = lp : rp
        If service candidate Sco_pos^m can satisfy sr for all the QoS
           minlostemp=slost(Sco_pos^m , sr);
           If minlosttemp< minlost
              minlost=minlosttemp;
              selectr=Sco_pos^m;
           Endif
        EndIf
   Endfor
9. If selectr=-1
        Assign selectr to sr;
   EndIf
      //minmore records the total value of the QoS of the web service that is more than the
requirement of the web service request under the condition that all of the hard requirements of
the service request have been satisfied.
10. pos = rp + 1;
      //search from the position of the end of the right position;
11. While (pos < ns_m + sum_m)
      11.1 If Sco_pos^m can satisfy the hard requirement of QoS of sr
              selectr=Sco_pos^m;
              Assign selectr to sr;
              Break;
           EndIf
   EndWhile
```

Algorithm 3: Schsoft (abstract service s_m).

$$wsr = \{ET_{wsr}, Rel_{wsr}, Av_{wsr}, Rep_{wsr}, Pr_{wsr}\} \tag{18}$$

$$hard_{wsr} = \{ETh_{wsr}, Relh_{wsr}, Avh_{wsr}, Reph_{wsr}, Prh_{wsr}\} \tag{19}$$

$$mvalue\left(QoS_{wsc}, QoS_{wsr}\right) = \frac{\sum more\left(QoS_{wsc}, QoS_{wsr}\right)}{moren} \tag{20}$$

$$more\left(ET_{wsc}, ET_{wsr}\right) = \begin{cases} +\infty, & if\ ET_{wsc} \le ET_{wsr}; \\ ET_{wsc} - ET_{wsr}, & if\ ET_{wsc} > ET_{wsr}; \end{cases} \tag{21}$$

$$more\left(Rel_{wsc}, Rel_{wsr}\right) = \begin{cases} +\infty, & if\ Rel_{wsc} \ge Rel_{wsr}; \\ Rel_{wsr} - Rel_{wsc}, & if\ Rel_{wsc} < Rel_{wsr}; \end{cases} \tag{22}$$

For the service requests which have not been assigned web services when we suppose every requirement of QoS attributes is hard, we will consider the soft requirement of QoS attributes. We add a new parameter to calculate the loss of the soft QoS attribute *(slost)* when a web service candidate cannot satisfy the requirement of the relaxable QoS attribute. Formula (23) is used to calculate the loss for the cost QoS attribute (execution time, similar to price) and formula (24) is used to calculate the loss for the benefit QoS attribute (reliability, similar to availability, and reputation). In fact, we

take the lost value as the QoE of the scheduling. Formula (25) gives the totally lost value in the soft QoS attribute. Formula (26) gives the method of calculating of average loss of soft requirements. $sofutnum$ is the number of the resources whose QoS attributes do not satisfy the requirement. The related formulas are as follows:

$$lost\left(ET_{wsc}, ET_{wsr}\right) = \begin{cases} 0, & if\ ET_{wcs} \le ET_{wsr}; \\ ET_{wsc} - ET_{wsr}, & if\ ET_{wsc} > ET_{wsr}; \end{cases} \tag{23}$$

$$lost\left(Rel_{wsc}, Rel_{wsr}\right) = \begin{cases} 0, & if\ Rel_{wsc} \ge Rel_{wsr}; \\ Rel_{wsr} - Rel_{wsc}, & if\ Rel_{wsc} < Rel_{wsr}; \end{cases} \tag{24}$$

$$\begin{aligned} lostvalue\left(wsc, wsr\right) = {} & lost\left(ET_{wsc}, ET_{wsr}\right) + lost\left(Rel_{wsc}, Rel_{wsr}\right) + lost\left(Av_{wsc}, Av_{wsr}\right) + lost\left(Rep_{wsc}, Rep_{wsr}\right) \\ & + lost\left(Pr_{wsc}, Pr_{wsr}\right) \end{aligned} \tag{25}$$

$$slost\left(wsc, wsr\right) = \begin{cases} +\infty, & if\ there\ are\ hard\ requirements\ cannot\ be\ satisified; \\ \dfrac{lostvalue}{sofutnum}, & if\ there\ are\ soft\ requirement\ cannot\ be\ satisfied; \end{cases} \tag{26}$$

The details of our scheduling algorithms are listed in Algorithms 2 and 3. Step 3.1 (in Algorithm 1) is the second step which is in charge of scheduling the abstract service s_m. "schhard(abstract service s_m)" (Algorithm 2) is the service composition method when we suppose every requirement of the QoS attribute is a hard requirement. First of all, we calculate the weight of different QoSs according to formulas in Section 5.1 (Step 1, in Algorithm 2), and calculate the aggregation value of the service candidate and the service request use the formulas in Section 5.2 (Step 2, in Algorithm 2). Then, we list all the values in a set *SCO*, and we sort the *SCO* in descending order. Three values are returned and they are sets *tp*, *ID*, and *Val*. "*tp*" has two possible values, if it equals "*candidate*"; it is a service candidate of the abstract service s_m; at this time, the value in *ID* is the service candidate and *Val* is the related value of aggregation function; if it equals "*request*", it is a service request to the abstract service s_m. At this time, the value of *ID* is the service request and *Val* is the related value of aggregation function. First, we will search the best fit service candidate between the left position *lp* and the right position *rp* (Step 6 -9, in Algorithm 2). *minmore* records the minimum value in the function of *mvalue* and *selectr* records the selected service candidates. If we can find some service candidates that can satisfy all the requirement when we suppose every requirement of the QoS attribute is a hard requirement, we will select the service candidate which has minimum value in the function of *mvalue* (Step 8, in Algorithm 2; formula (20)). If we cannot get a service candidate between the left position and the right position, we will search from the end of the right position until we find a service candidate which can satisfy all the requirements of the QoS under the assumption that every requirements of the QoS attribute which belongs to the hard requirement (Step 11, in Algorithm 2) have been satisfied.

Algorithm 3 is the service selection method for the soft requirement of QoS attributes under the method of GLLB. We will first search the best fit service candidate between the left position *lp* and the right position *rp* (Steps 6-9, in Algorithm 3). *minlost* records the minimum value in the function of *slost* (Formula (26)) and *selectr* records the selected service candidate. If we can find some service candidates that can satisfy all the requirement when we suppose every requirement of the QoS attribute is a hard requirement, we will select the service candidate which has minimum value in the function of *slost* (Step 8, Algorithm 3; formula (26)). If we cannot get a service candidate between the left position and the right position which can satisfy the hard requirement of the QoS attribute, we will search from the end of the right position until we find a service candidate which can satisfy all the requirement of the QoS, under the supposition that we can give up some soft requirements (Step 11, Algorithm 3).

5.4. Complexity Analysis of GLLB. We analyze our method for one special abstract service s_m from three steps:

(1) Calculating the weight of different QoSs (formulas in Section 5). The number of the candidate services is sum_m, the number of service requests is num_m, and we only pay attention to five QoS attributes. So, the complexity is $O(5 * num_m * sum_m)$.

(2) Computing the normalized quality values of all service candidates and service requests. The complexity of calculating average value is $O(num_m * sum_m)$, and the complexity of calculating standard deviation is also $O(num_m * sum_m)$. We take account of five QoS attributes, so the complexity is $O(5 * num_m * sum_m)$.

(3) The core of GLLB (Algorithms 2 and 3). Under the worst case, every service request will check every service candidate, so the complexity is $O(num_m * sum_m)$.

Table 1: Values of five QoSs.

QOS	QoS Range	QoS Constraint
Execution time	[1,100]	[1+99∗*CF*,100]
Reliability	[0.95,0.9999]	[0.95,0.9999-0.0499∗*CF*]
Availability	[0.95,0.9999]	[0.95, 0.9999-0.0499∗*CF*]
Reputation	[1,10]	[1, 10-9∗*CF*]
Price	[1,100]	[1+99∗ *CF*,100]

And thus, in generally, the complexity of our method is

$$\begin{aligned} &O(5 * num_m * sum_m) + O(5 * num_m * sum_m) \\ &\quad + O(num_m * sum_m) = O(num_m * sum_m) \end{aligned} \tag{27}$$

For all the abstract services, suppose the total number of the abstract services is M; the complexity is $O(M * num_m * sum_m)$.

6. Experiment Results and Discussions

In the simulation, we will compare our method to RQSS [8], MDSC (Multidimension Service Composition) [28], and RASC (Rank Aggregation Service Composition) [32]. RQSS [8] helps user discover feasible web services based on functionalities and QoS requirements. RQSS recommends prospective service candidates to users by relaxing QoS constraints if no available web service could exactly fulfill users' requirements. MDSC [28] first normalizes those QoS parameters and then uses the improved Euclidean Distance to select the web service which meets the requirement (in the simulation, we suppose every part has the same weight). RASC [32] uses Rank Aggregation methods to solve the web service composition.

Although there are many kinds of structures of the service composition, such as the cycle, sequence structure, parallel structure, loop structure, and case structure, we just pay attention to the sequence structure. All other structures can be transformed into sequence structures by some methods [28, 29].

6.1. Simulation Environment. We utilize the Matlab language to implement these algorithms and ran them on an Intel Pentium (R) D 3.4 Ghz, 4 GB RAM desktop PC with 100 MB/s Ethernet card, Window 8. In the simulation, we consider five QoS attributes which have been discussed before: execution time, reliability, availability, reputation, and price. The value of each QoS attribute is randomly generated with a uniform distribution in a range, as shown in Algorithm 3. These ranges are also used in some researches [28]. In the simulation, we assume that the execution times of all services are less than 100 *ms*. And most of services are robust enough. Therefore, the reliability and availability of service are ranging from 0.95 to 0.99999. Those settings are reasonable for web services [32]. In addition, the reputation has range of [1, 10] and the price has range of [1, 100]. These assumptions are also used in [8] and they are reasonable in various applications and different environments.

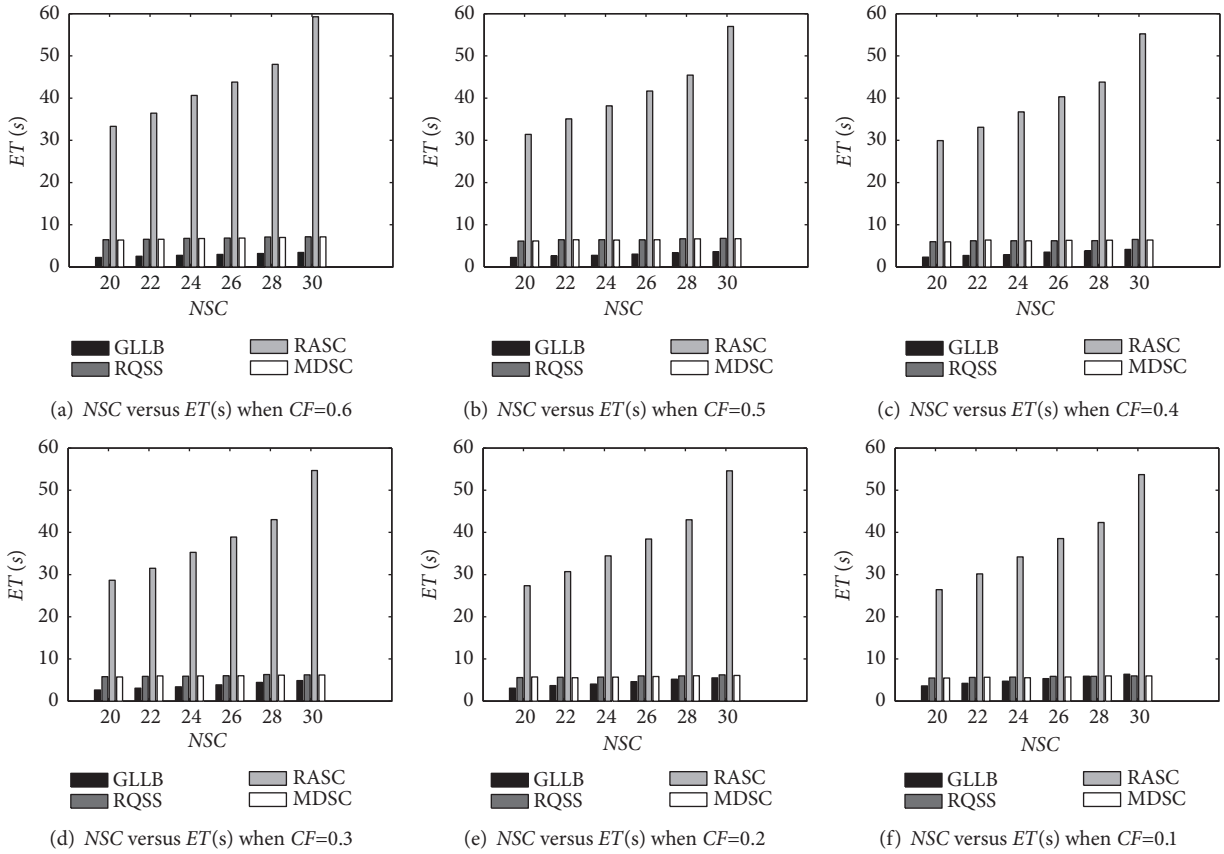
The QoS constraint is generated as the last column in Table 1. The method also was used in [28], but there is some difference. *CF* is the constraint factor and it has a range of [0, 1]. The values of relative QoS attributes are random in the range of the last column in Table 1 and they follow the uniform distribution. It can be used to represent the strength of a QoS constraint. If a requirement of the QoS is a soft one: a lower value of *CF* always means a lower of the QoS constraint (the QoE is same to the Figure 2(b) or Figure 2(f)). When the QoS is in the scope of the last column of Table 1, the QoE of relative QoS is 1. If a requirement of the QoS is a hard one, the QoE is the same to Figure 2(c) or Figure 2(g). If the QoS is not in the scope of the last column of Table 1, the QoE would be 0; otherwise, it is 1.

We will give comparisons from the execution time (*ET*), the number of the unfinished web service requests (*NUR*), the total value of failure to QoS attributes (*FV*), and the average failure to the QoS attributes (*AFV*). There are 500 abstract services and 2000 service processes to those abstract services. The possibility of a service process including an abstract service is 1%. We will evaluate the value of *CF* changed from 0.6 to 0.1 with a step of -0.1. The number of service candidates to every abstract service is changed from 20 to 30 with a step of 2. Every QoS has 75% possibility that it belongs to the hard requirement. The evaluated values in the following section are the average values of 50 times executions.

6.2. Execution Time. Figure 3 is the execution times of different methods under various numbers of service candidates and diverse values of *CF*. The values of X axes are NSC (number of service candidates) and the values of Y axis are execution times. All methods have the same trends: *ET* increases with the increasing of *CF* and the number of service candidates. The reason is that, with the increasing of *CF*, the service request needs to check more service candidates, and with the increasing of the number of the service candidates, the service request has more chances to select.

In general, RASC always has the largest value in *ET* and its average value under all the conditions is 39.8603 (s); GLLB always has the lowest value in *ET* and its average value is 3.6867 (s). GLLB only has 9.25% execution time to RASC. To RQSS and MDSC, the GLLB average value is reduced by 59.59% and 59.70%, respectively.

RASC has the largest value in *ET*, the reason is the calculations of the Kendall Tau Distance between the service request and the service candidates needs much time [32]. The execution time is a polynomial function to the number of service candidates. RQSS and MDSC do not lead to any difference in the execution time, and the execution times



(a) *NSC* versus *ET*(s) when *CF*=0.6 (b) *NSC* versus *ET*(s) when *CF*=0.5 (c) *NSC* versus *ET*(s) when *CF*=0.4

(d) *NSC* versus *ET*(s) when *CF*=0.3 (e) *NSC* versus *ET*(s) when *CF*=0.2 (f) *NSC* versus *ET*(s) when *CF*=0.1

Figure 3: The execution time of different methods.

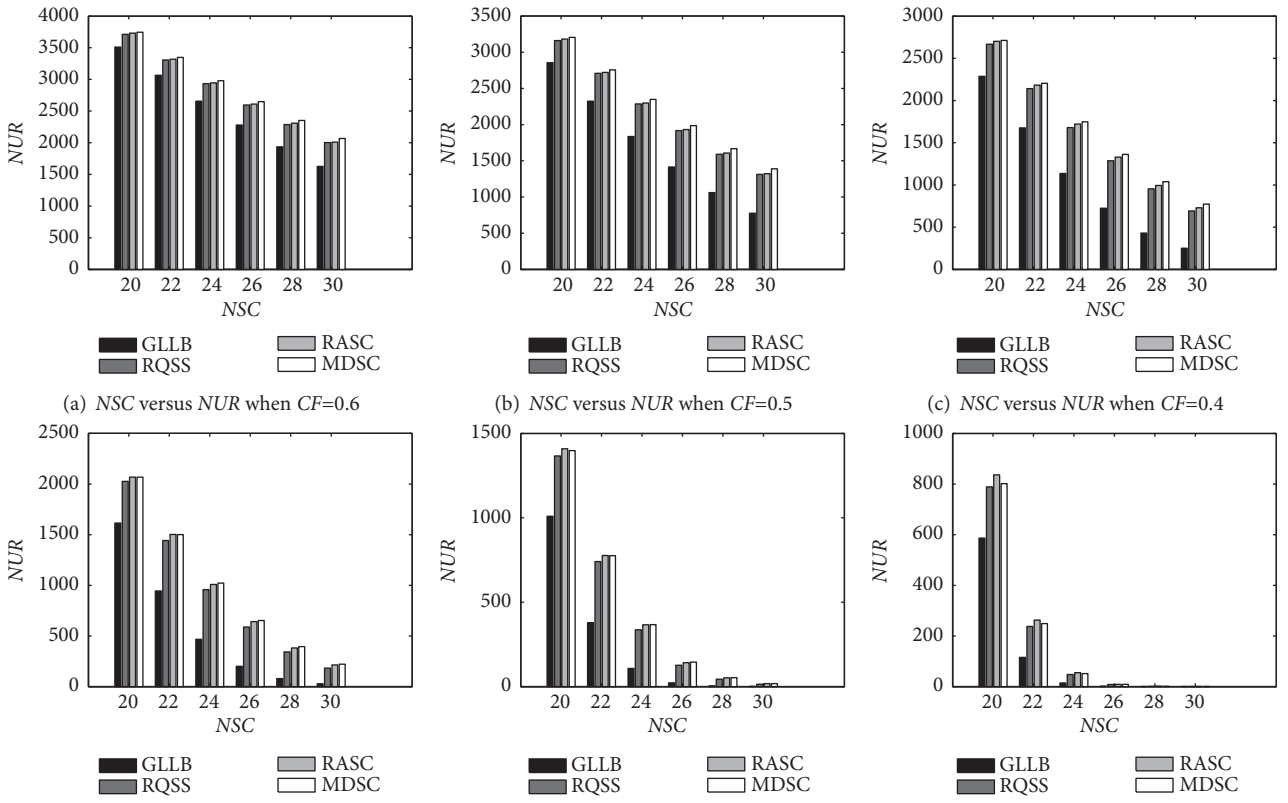
of them are a linear function to the number of the service candidates. The ranking of GLLB reduces the time to search all the service candidates. Most of time, GLLB only needs searching between the left and the right of the service request. By the way, in the simulation, we may spend some time to get the normalize value (Section 5.1), but we can get in before scheduling, so the execution time of GLLB does not include the time to get the relative normalize time. A tradeoff way is to get those data (average value; standard deviation) from the past record.

6.3. Comparison of the Number of Unfinished Web Service Requests in Finding a Feasible Composition. Figure 4 shows the number of unfinished service requests (*NUR*) of the four methods under different numbers of service candidates and various values of *CF*. The values of X axes are the number of service candidates and the values of Y axes are the values of *NUR*. The four methods involved in the comparison have the same patterns: *NUR* drops with the reducing of the number of service candidates and the value of *CF*. The reasons are as follows: with the dropping of the number of service candidates, there are fewer service candidates for the service request; at the same time, with the dropping of *CF*, the stronger requirements of QoSs are made, and the more service requests cannot get the required services. GLLB always has the lowest value in *NUR* and it has an average value of 1039. The other three methods have the same performance in *NUR* basically. The average *NUR* of RQSS, RASC, and MDSC is 1346, 1372, and 1390, respectively. To RQSS, RASC, and MDSC, the GLLB average id reduced by 22.81%, 24.27%, and 25.25%, respectively.

GLLB has the lowest value in *NUR* because of the policy: when we suppose, every requirement is a hard requirement, we try to find the service candidate which has the nearest value to all the QoS, the policy makes the best fit service candidate always be selected first, and it makes the left service requests have more opportunities to get the required service candidates.

6.4. Comparison of Average Violated Quality Value (AVQV) to the QoS Requirement. Figure 5 shows the failure value to the QoS requirement (*FV*) of the four methods under different numbers of service candidates and different values of *CF*. The values of X axes are the number of service candidates and the value of Y axis are the value of *FV*. The four methods behave same patterns: they all drop with the reducing of the number of service candidates and the value of *CF*.

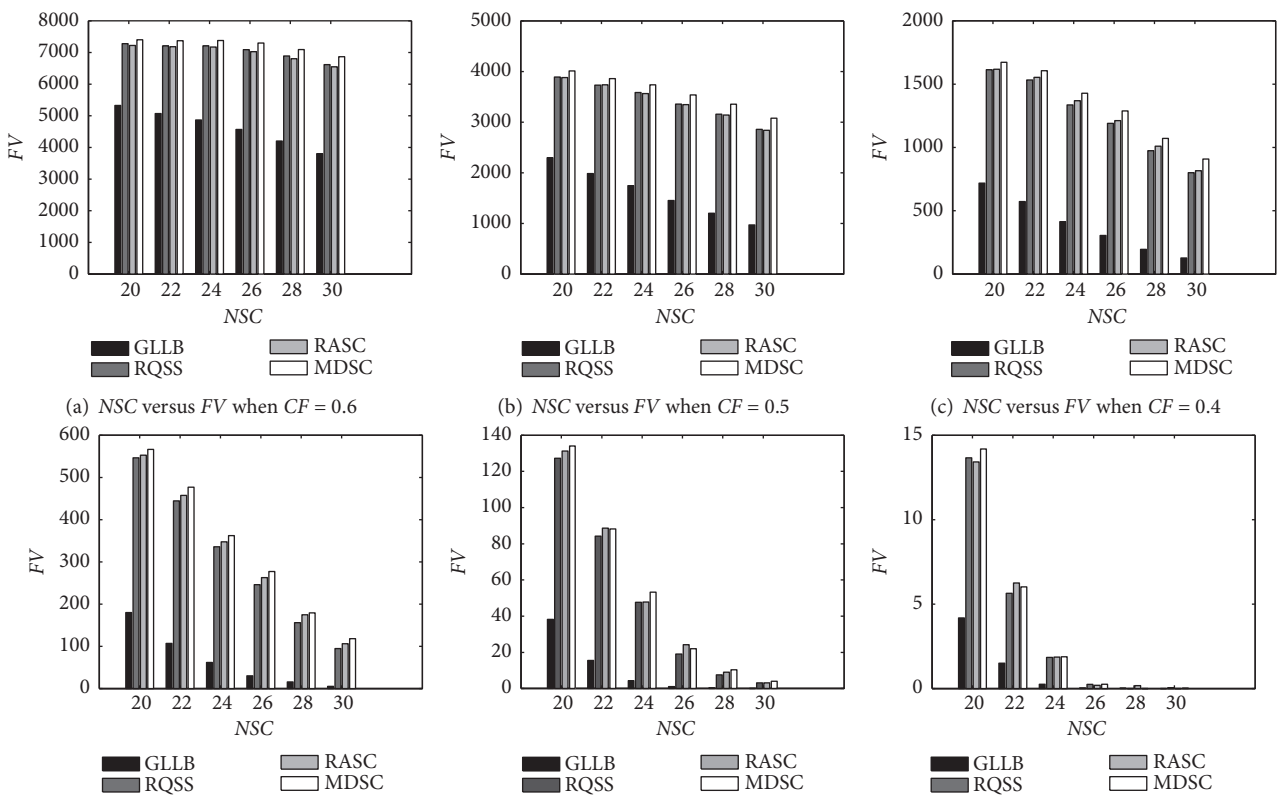
GLLB always has the lowest value in *FV* and it has an average value of 1190. The other three methods do not show much different performance in *FV*. The average *FV* of RQSS, RASC, and MDSC is 2012, 2007, and 2091, respectively. To RQSS, RASC, and MDSC, the average value of FV in the case of using GLLB is enhanced by 79.80%, 79.36%, and 86.86%, respectively.



(a) *NSC* versus *NUR* when *CF*=0.6 (b) *NSC* versus *NUR* when *CF*=0.5 (c) *NSC* versus *NUR* when *CF*=0.4

(d) *NSC* versus *NUR* when *CF*=0.3 (e) *NSC* versus *NUR* when *CF*=0.2 (f) *NSC* versus *NUR* when *CF*=0.1

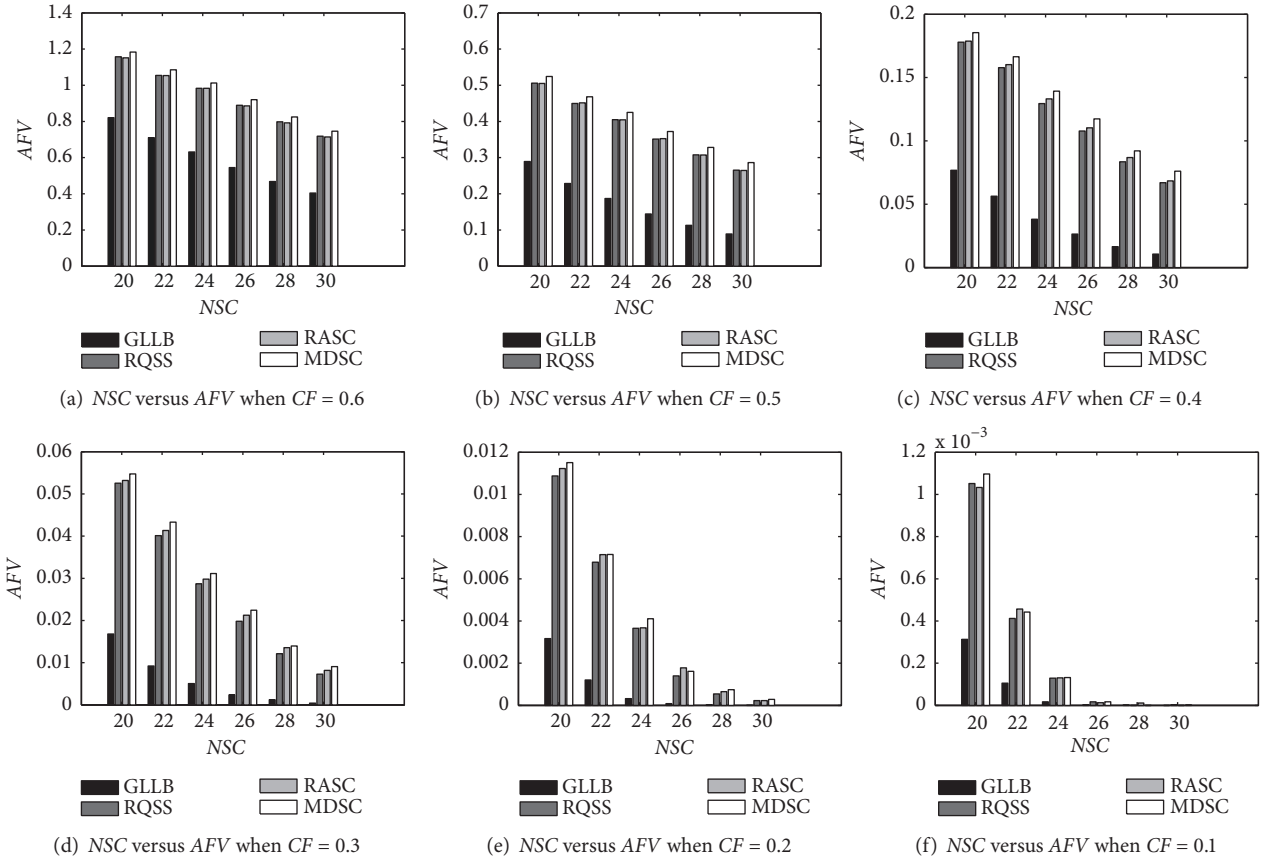
Figure 4: The number of unfinished web service requests.



(a) *NSC* versus *FV* when *CF* = 0.6 (b) *NSC* versus *FV* when *CF* = 0.5 (c) *NSC* versus *FV* when *CF* = 0.4

(d) *NSC* versus *FV* when *CF* = 0.3 (e) *NSC* versus *FV* when *CF* = 0.2 (f) *NSC* versus *FV* when *CF* = 0.1

Figure 5: The total Failure value to the soft QoSs of different methods.



(a) *NSC* versus *AFV* when *CF* = 0.6
(b) *NSC* versus *AFV* when *CF* = 0.5
(c) *NSC* versus *AFV* when *CF* = 0.4
(d) *NSC* versus *AFV* when *CF* = 0.3
(e) *NSC* versus *AFV* when *CF* = 0.2
(f) *NSC* versus *AFV* when *CF* = 0.1

Figure 6: The Average failure value to the soft QoSs of different methods.

GLLB has the lowest value in *FV* because: when we suppose all the requirement of the QoSs are hard, we try to find the service candidates which are the nearest to the service candidates; this makes other service requests have more chances to get the right service candidate which can satisfy all the requirement; no matter they belong to the hard requirement or the soft requirement; if GLLB finds that a service process cannot get the right service candidate in the first step, we always try to find the service candidate when the scheduling has the lowest value in *FV*.

Figure 6 shows the average failure value to the QoS requirement (*AFV*) of the four methods under different numbers of service candidates and different value of *CF*. The value of X axis is the number of service candidates and the value of Y axis is the value of *AFV*. The four methods show the same patterns: they all have a dropping tendency when the number of service candidates and the value of *CF* are reducing.

GLLB always has the lowest value in *AFV* and it has an average value of 0.1360. The other three methods do not show much different performance in *AFV*. The average *AFV* of RQSS, RASC, and MDSC is 0.2443, 0.2444, and 0.2543, respectively. *AFV* of GLLB is only about 55.67%, 55.65%, and 53.48% to the *AFV* of RQSS, RASC, and MDSC.

GLLB has the lowest value in *AFV* because it has the lowest value in *FV* (Figure 5) and it also has the lowest value in *NUR* (Figure 4).

7. Conclusion

In the paper, we analyze the service composition problem from the view of QoS and QoE. Then, we rank the service composition requests and the services in order. Based on the rank of them, a service selection and composition method is proposed to meet the environment when there are soft and hard requirements of the QoS attributes of web services. First of all, we take all the requirements of QoS attributes as the hard requirement, if we cannot get the right service candidate; we try to find service candidate by relaxing some QoSs under the permission of the service request.

In the Cloud, when there are many Clouds and every Cloud has various kinds of abstract services and different numbers of related abstract services, the service composition problem becomes more difficult. Different QoSs have diverse "overload" values [33]. For the multiple Clouds environment [34], more services and more choices bring more challenge to service composition. We not only want to consider the requirement of every web service, but also want to reduce the number of the related Clouds that involved in the service composition. The problem of service composition under multiple Clouds [34] is the future work of us. Because we need to normalize the QoS, we may optimize our method to ensure them work well just from a small local scope and then work for overall situation. In this paper, we just evaluate our work on a simulation environment; as a future work, we also hope

we can evaluate our work on a real web service composition case.

Data Availability

Below is the weblink where other researchers can download the data: https://pan.baidu.com/s/12NjYa8q5wrV4AmcyVc7XYA.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work was partly supported by the National Natural Science Foundation of China (NSF) under Grant no. 41475089 and major projects of the National Social Science Fund (16ZDA054), the Natural Science Foundation of Zhejiang Province (LQ18F020005), and Science and Technology of Wenzhou (S20170008).

References

[1] Y. Hao, L. Wang, and M. Zheng, "An adaptive algorithm for scheduling parallel jobs in meteorological Cloud," *Knowledge-Based Systems*, vol. 98, pp. 226–240, 2016.

[2] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1012–1023, 2013.

[3] A. Ramírez, J. R. Romero, A. Ruiz-Cortés et al., "Evolutionary composition of QoS-aware web services: A many-objective perspective," *Expert Systems with Applications*, vol. 72, pp. 357–370, 2017.

[4] V. Gabrel, M. Manouvrier, K. Moreau, and C. Murat, "QoS-aware automatic syntactic service composition problem: Complexity and resolution," *Future Generation Computer Systems*, vol. 80, pp. 311–321, 2018.

[5] H. Kurdi, A. Al-Anazi, C. Campbell, and A. A. Faries, "A combinatorial optimization algorithm for multiple Cloud service composition," *Computers & Electrical Engineering*, vol. 42, pp. 107–113, 2015.

[6] B. Upadhyaya, Y. Zou, J. Ng, T. Ng, and D. Lau, "Towards quality of experience driven service composition," in *Proceedings of the 2014 IEEE World Congress on Services (SERVICES)*, pp. 18–20, Anchorage, AK, USA, June 2014.

[7] 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.

[8] C.-F. Lin, R.-K. Sheu, Y.-S. Chang, and S.-M. Yuan, "A flexibleservice selection algorithm for QoS-based web service composition," *Information and Software Technology*, vol. 53, no. 12, pp. 1370–1381, 2011.

[9] N. Anithadevi and M. Sundarambal, "A design of intelligent QoS aware web service recommendation system," *Cluster Computing*, vol. 1, pp. 1–10, 2018.

[10] Y. Feng, L. D. Ngan, and R. Kanagasabai, "Dynamic service composition with service-dependent QoS attributes," in *Proceedings of the 2013 IEEE 20th International Conference on Web Services, ICWS 2013*, pp. 10–17, USA, July 2013.

[11] R. Iordache and F. Moldoveanu, "A web service composition approach based on QoS preferences," in *Proceedings of the 6th IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2013*, pp. 220–224, USA, December 2013.

[12] G. Liu, Y. Wang, and M. A. Orgun, "Finding K optimal social trust paths for the selection of trustworthy service providers in complex social networks," in *Proceedings of the 2011 IEEE 9th International Conference on Web Services, ICWS 2011*, pp. 41–48, USA, July 2011.

[13] L. Li, Y. Wang, and E. P. Lim, "Trust-oriented composite service selection with QoS constraints," *Journal of Universal Computer Science*, vol. 16, no. 13, 2010.

[14] T. Hofeld, L. Skorin-Kapov, P. E. Heegaard et al., "A new QoE fairness index for QoE management," *Quality & User Experience*, vol. 3, no. 1, p. 4, 2018.

[15] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu, "Web service recommendation via exploiting location and QoS information," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1913–1924, 2014.

[16] Q. Yu, "CloudRec: a framework for personalized service Recommendation in the Cloud," *Knowledge and Information Systems*, vol. 43, no. 2, pp. 417–443, 2015.

[17] W. Denghui, H. Hao, and X. Changsheng, "A novel web service composition recommendation approach based on reliable QoS," in *Proceedings of the IEEE 8th International Conference on Networking, Architecture and Storage (NAS '13)*, pp. 321–325, IEEE, Xi'an, China, July 2013.

[18] S. Zhang, W. Dou, and J. Chen, "Selecting top-k composite web services using preference-aware dominance relationship," in *Proceedings of the 2013 IEEE 20th International Conference on Web Services, ICWS 2013*, pp. 75–82, USA, July 2013.

[19] H. Wang, Y. Tao, Q. Yu et al., "Incorporating both qualitative and quantitative preferences for service recommendation," *Journal of Parallel & Distributed Computing*, vol. 114, 2017.

[20] Y. Yu, H. Ma, and M. Zhang, "A Genetic Programming approach to distributed QoS-aware web service composition," in *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, pp. 1840–1846, China, July 2014.

[21] Z. Ding, J. Liu, Y. Sun, C. Jiang, and M. Zhou, "A transaction and QoS-aware service selection approach based on genetic algorithm," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 7, pp. 1035–1046, 2015.

[22] M. Cremene, M. Suciu, D. Pallez, and D. Dumitrescu, "Comparative analysis of multi-objective evolutionary algorithms for QoS-aware web service composition," *Applied Soft Computing*, vol. 39, pp. 124–139, 2016.

[23] Q. Yu, L. Chen, and B. Li, "Ant colony optimization applied to web service compositions in cloud computing," *Computers & Electrical Engineering*, vol. 41, pp. 18–27, 2015.

[24] S. Bharathan, C. Rajendran, and R. P. Sundarraj, "Penalty based mathematical models for web service composition in a geo-distributed cloud environment," in *Proceedings of the 24th IEEE International Conference on Web Services, ICWS 2017*, pp. 886–889, USA, June 2017.

[25] W. Chen, I. Paik, and P. C. K. Hung, "Constructing a global social service network for better quality of web service discovery," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 284–298, 2015.

[26] S. Chakhar, "QoS-enhanced broker for composite web service selection," in *Proceedings of the 8th International Conference on*

Signal Image Technology and Internet Based Systems, SITIS 2012, pp. 533–540, Italy, November 2012.

[27] A. ShaikhAli, O. F. Rana, R. Al-Ali, and D. W. Walker, "UDDIe: An extended registry for Web services," in *Proceedings of the 2003 Symposium on Applications and the Internet Workshops, SAINT 2003*, pp. 85–89, USA, January 2003.

[28] L. Li, M. Rong, and G. Zhang, "A web service composition selection approach based on multi-dimension QoS," in *Proceedings of the 8th International Conference on Computer Science and Education, ICCSE 2013*, pp. 1463–1468, Sri Lanka, August 2013.

[29] L. Y. Qi, Y. Tang, W. C. Dou, and J. J. Chen, "Combining local optimization and enumeration for QoS-aware web service composition," in *Proceedings of the IEEE International Conference on Web Services*, pp. 34–41, July 2010.

[30] S. Wang, C.-H. Hsu, Z. Liang, Q. Sun, and F. Yang, "Multi-user web service selection based on multi-QoS prediction," *Information Systems Frontiers*, vol. 16, no. 1, pp. 143–152, 2014.

[31] M. Ortega, Y. Rui, K. Chakrabarti, S. Mehrotra, and T. S. Huang, "Supporting similarity queries in MARS," in *Proceedings of the 1997 5th ACM International Multimedia Conference*, pp. 403–413, November 1997.

[32] B. Hofreiter and S. Marchand-Maillet, "Rank aggregation for QoS-aware Web service selection and composition," in *Proceedings of the 6th IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2013*, pp. 252–259, USA, December 2013.

[33] Y. Hao, "Enhanced resource scheduling in Grid considering overload of different attributes," *KSII Transactions on Internet and Information Systems*, vol. 10, no. 3, pp. 1071–1090, 2016.

[34] Y. Hao, M. Xia, N. Wen et al., "Parallel task scheduling under multi-clouds," *KSII Transactions on Internet and Information Systems*, vol. 11, no. 1, pp. 39–60, 2017.