# A Dynamic QoS Adjustment Enabled and Load-balancing-aware Service Composition Method for Multiple Requests

**Xiaozhu Wu***
College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China
Key Lab of Spatial Data Mining & Information Sharing, MOE, Fuzhou University, Fuzhou 350108, China
[e-mail: wxz@fzu.edu.cn]
*Corresponding author: Xiaozhu Wu

*Received October 26, 2020; revised December 15, 2020; accepted February 11, 2021; published March 31, 2021*

## Abstract

Previous QoS-aware service composition methods mainly focus on how to generate composite service with the optimal QoS efficiently for a single request. However, in the real application scenarios, there are multiple service requests and multiple service providers. It is more important to compose services with suboptimal QoS and maintain the load balance between services. To solve this problem, in this paper, we propose a service composition method, named as dynamically change and balancing composition method (DCBC). It assumes that the QoS of service is not static, and the services can adjust the value of QoS to gain more opportunities to be selected for composition. The method mainly includes two steps, which are the preprocessing step and the service selection step. In the preprocessing step, a backward global best QoS calculation is performed which regarding the static and dynamic QoS respectively; then guided by the global QoS, the feasible services can be selected efficiently in the service selection step. The experiments show that the DCBC method can not only improve the overall quality of composite services but also guarantee the fulfill ratio of requests and the load balance of services.

***Keywords:*** Dynamic QoS, Load Balance, Service Composition, QoS Adjustment

This research was supported by Key Science and Technology Plan Projects of Fujian Province (2015H0015), Education and Technology Plan Projects of Fujian Province (JAT160088), and Foundation of China Scholarship Council (201706655035).

**http://doi.org/10.3837/tiis.2021.03.005** **ISSN : 1976-7277**

## 1. Introduction

Web service is a widely used technology to enable business cooperation nowadays [1]. Many enterprises have published a lot of web services to seek new business opportunities, such as Google, Facebook and Amazon, which could be found in the online web service repository [2]. The web service resources provide the potential to integrate two or more services to fulfill the user's complicate requirement. Service composition also is a promising technology to realize the business integration. Service composition algorithms try to select appropriate services to form a service chain or workflow so as to satisfy the user's functional and non-functional constraints [3].

The QoS-aware web service composition is one of the most popular research topics in the field of service computing [4-8]. The QoS attributes usually refer to the non-functional characteristics of web service, such as availability, reliability, response time and cost. The QoS-aware web service composition mainly focuses on how to select component services to get optimal overall QoS in a cost-effective manner. Due to the exploration of solution space, it is a time-consuming process to pick up candidate service. Many heuristic algorithms have been proposed to tackle this problem [9-12]. However, most of the current researches always assume that the value of the QoS attribute is static and cannot be changed in the runtime [13, 14]. But this may not be true in the competitive business society. For instance, a cloud service provider is willing to expand the network bandwidth or cut down the price to improve its competitiveness at the initial stage. Then, the provider will raise the price to ensure the profit growing when he already occupied the majority of the market share. Hence the service composition method needs to consider the proactive change of the QoS.

Furthermore, most of the works tend to generate the composite service with the optimal QoS, hence only a few services that have the best QoS can be selected. But this strategy may be not applicable in practice: 1) there is only a small part of services that can be chosen to provide service for users, which makes these services suffer heavy load burden. Furthermore, these services may be unavailable if the workloads exceed their maximum capacity; 2) some services may have little chance to be invoked, even if their QoS can meet the user's requirement. This will lead to the load-unbalance between services. It is not true that every user wants to request the best service. Some users may choose the service with medium QoS due to their limited budget. For example, some tourists may book five-star hotels, but others may prefer motels or other hotels at a lower price. Therefore, it is not necessary to generate the best composite service for every client. The different service providers with diverse QoS shall have an equal chance to provide service for users.

In this paper, we emphasize the problem of QoS-aware service composition in the multiple request environment, which takes the dynamic QoS adjustment and service load-balance into account. This research aims to match the requests with the appropriate services and make sure that 1) most of the requests can be satisfied; 2) the quality of all of the generated composite services shall be optimal as far as possible; 3) the workload of services shall be balanced. To tackle this problem, we propose a novel service composition method named as DCBC (dynamically change and balancing composition) which considers the QoS and load-balance simultaneously. This method proposes a mechanism to allow the service provider to adjust the value of QoS attributes dynamically to improve the probability to be selected. Meanwhile, the method tries to maintain the relative load balance between services. In this method, the services with relative fewer requests will be encouraged to improve their QoS to gain more opportunities to participate in composition.

The rest of the paper is organized as follows: Section 2 introduces a motivation example. Section 3 reviews the related works. The definition of related concepts and the problem statement are presented in the section 4. Details of the DCBC method are presented in the section 5. The experiments and results are discussed in the section 6. The section 7 draws the conclusions.

## 2. Motivation Example

In this section, we introduce a typical example in the field of collaborative cloud manufacturing to demonstrate how the service providers can collaborate and compete with each other to fulfill the client's request and keep the load-balance. **Fig. 1** shows a simple product manufacturing workflow, where the rectangle represents the virtual manufacturing service and the circle represents the concrete manufacturing service. This workflow consists of three virtual services $(VS_1, VS_2, VS_3)$, which indicates that the product contains three components and they need to be produced by three independent manufacturers. Each virtual service has several candidate concrete services, which can produce the required components of the final product. For example, $VS_1$ has three candidate concrete services $(s_{11}, s_{12}, s_{13})$, which are published by different factories. The detail information of candidate concrete services is presented in **Table 1**. For simplicity, in this example, we only consider one QoS attribute. In **Table 1**, the PRICE column shows the price to invoke the service to produce a component product, and the DISCOUNT column shows the potential price reduction space, which is affected by the market competition and decided by the service owner. For instance, although $s_{11}$'s original price is 100, but the service owner can adjust the price to be 90 with 10% discount if it wants to improve the competitiveness. In contrast, $s_{13}$ will always keep its price unchanged because the discount value is set as NA.

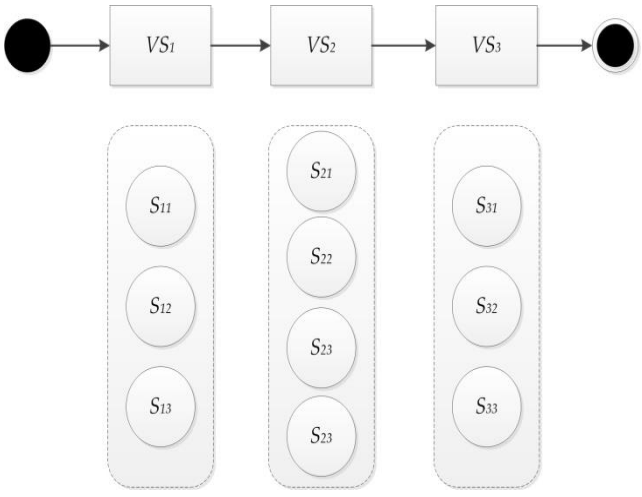


**Fig. 1.** Service composition example

Suppose that there are five requests in a queue launched by clients: $\{r_1 = 180, r_2 = 200, r_3 = 130, r_4 = 150, r_5 = 138\}$, where the value denotes the maximum price that the client can afford. Hence the requirement of the request is that the total price of the manufacturing workflow must not exceed the limit. We assume that the requests are independent and arrive at a different time, and they will be served in a FIFO mechanism.

**Table 1.** The Details of Candidate Concrete Services in the Example

| SID | PRICE | DISCOUNT |
|---|---|---|
| $S_{11}$ | 100 | 10% |
| $S_{12}$ | 90 | 20% |
| $S_{13}$ | 80 | NA |
| $S_{21}$ | 40 | 20% |
| $S_{22}$ | 50 | 10% |
| $S_{23}$ | 60 | NA |
| $S_{24}$ | 30 | NA |
| $S_{31}$ | 28 | NA |
| $S_{32}$ | 30 | 20% |
| $S_{33}$ | 27 | NA |

The traditional optimal service composition methods always try to find the best solutions while ignoring the balance between service providers [15]. Without considering the variability of QoS, to deal with each of the five requests, the traditional optimal methods will always generate the composite service $\{s_{13}, s_{24}, s_{33}\}$ with the total price of 137. Obviously, four requests will be fulfilled except for $r_3$ in that the price constraint of $r_3$ is lower than the total price of any composition. However, these solutions may not be optimal in practice. From the point view of service providers, the service group $\{s_{13}, s_{24}, s_{33}\}$ will be invoked four times, while the others will be idle, which leads to load imbalance. For clients, it is unreasonable to provide composite service with low price to who could afford the high cost, because that the lower price always means lower quality in other QoS attributes. A possible solution for the example is shown in **Table 2**. Each of the rows shows the selected concrete services for each request. The last column TP shows the total price of the generated composite service. The last row presents the load of each concrete service. It is easy to find out that although the total price of composite services generated by this solution is higher than the previous one, it can achieve a better balance between services. It must be noted that this result is not unique and the best one. Our goal is to generate composite services with better QoS and try to maintain a better balance between services.

**Table 2.** A Possible Solution for the Example

| REQ | $S_{11}$ | $S_{12}$ | $S_{13}$ | $S_{21}$ | $S_{22}$ | $S_{23}$ | $S_{24}$ | $S_{31}$ | $S_{32}$ | $S_{33}$ | TP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_1$=180 | | √ | | | √ | | | | | √ | 167 |
| $r_2$=200 | √ | | | | | √ | | | √ | | 190 |
| $r_3$=130 | | | | | | | | | | | NA |
| $r_4$=175 | | | √ | √ | | | | √ | | | 148 |
| $r_5$=138 | | | √ | | | | √ | | | √ | 137 |
| load | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | |

Furthermore, to satisfy more clients' requirement and gain more profits, service providers are willing to adjust the price of service. In this example, request $r_3$ cannot be fulfilled at the initial price setting of services. But if the price discount is taken into account, a composite service $\{s_{12}(20\%), s_{24}, s_{32}(20\%)\}$ with the total price, 126 can be generated to fulfill the request $r_3$, where service $s_{12}$ and $s_{32}$ both provide a 20% discount. In that case, all of the requests can be served.

The example reveals that, in order to maintain the load balance between services, it is more important to generate suitable composite service rather than the optimal one. The suitable composition may not be the best one according to QoS but must be the one that meets the user's constraint. In a real application, the QoS of service may change over time. Therefore, the composite service generated by the methods which considering the variable QoS can be better than the ones produced by traditional composition methods.

Different from the traditional service composition problem which aims to find the optimal solution, there may be multiple candidate services can be selected to fulfill a request. Consequently, it is more difficult to select services with better QoS while keeping the load balance. When considering how to embed the QoS adjustment strategy into the process of service composition to improve the quality of the solution, the problem becomes more complicate.

## 3. Related Work

The literature has presented several pieces of research about QoS-aware service composition which focus on the dynamic QoS or load-balance composition respectively. Chen et al. proposed an event-driven continuous query algorithm to deal with the dynamic service [16]. The dynamic situations include the unavailability of service, the change of QoS and the change of interface. A dependency graph was built in the paper to facilitate the searching for substitution services. Once a dynamic event occurred, the method would conduct a forward search starting from the dynamic service and its successors that may be affected. This method could achieve better performance because only the region of the affected services was updated instead of all of the web services. The other similar researches also focused on the problem of adaptive service composition when the QoS of some component services changed [17-20]. In these researches, the mixed integer programming and heuristic algorithm was used to select alternate services for the failure services or the services with decreased QoS value. Although the above- described works have identified the fact that the QoS is not static in real scenarios, they differ with our work in that: 1) the time when to execute the algorithm is different. The methods proposed by the above papers are reactive and to be executed when the event of service failure or QoS decreasing happened, but our method is executed proactively to create new composite services; 2) the purpose is different. These methods mainly try to keep the availability or the optimality of the composite service, but the target of our method is to satisfy the requests and keep the load-balance of services; 3) the effect is different. These methods could only take care of one single composite service at a time and do not account for the balance of services, but our method can handle multiple composite services and keep the load-balance between them.

Some researchers have focused on the problem of the uneven distribution of service requests in the field of cloud computing [21, 22]. Borrowing from the idea of the traditional load balancing strategy, some researches have been carried out to focus on the load-balancing service composition. In the background of collaborative manufacturing, Xue et al. proposed a computational experiment-based method to evaluate the effect of non-equalization strategy, equalization strategy and collaborative equalization strategy on the non-equalization phenomenon [23]. The differences between the three strategies lay in the QoS model, candidate service solutions set and the adopted composition algorithms. The experiment showed that the equalization strategy which considered the changeable QoS and the horizontal collaboration can enhance the utilization rate of most of the services in oversupply market environment. However, this research does not present a specific algorithm to deal with

equalization-oriented service.

Wang et al. proposed a 0-1 integer programming based service selection method that considered the service quality and load balance simultaneously [24]. In this study, the Prospect Theory was adopted to build the QoS utility model; a service load capacity computation model was also presented. Then they used the 0-1 IP algorithm to select services for multiple requesters and achieve the maximum value of the combination of the QoS utility and load capacity. Kang et al. proposed a similar method to maximize the matching degree between the requests and the web services selected under an environment of multiple requests and web services [25]. In this research, the Euclidean distance with the weight measurement method was utilized to evaluate the similarity of the request and service. The 0-1 integer programming algorithm enhanced with Skyline dominant relation was proposed to produce global optimal web service selection and keep the load balance between services. However, the studies described above were designed to resolve the problem of atomic service selection, which can be considered as the simplified form of the service composition problem. So the methods proposed in these researches are not suitable for the problem of multiple service composition with load balance.

Some researches treat the problem of multi-user web service selection as the maximum matching problem in the bipartite graph and its variants [26-28]. In paper [26], based on the unbalance transportation problem, Adrian et al. use the Vogel Approximation Method and Transportation Simplex Method to map the requests to concrete services. Similarly, Wang et al. proposed a Kuhn-Munkres algorithm-based method to match the requests with services and prevent the overload of services [27]. In paper [28], Jin et al. proposed a hybrid method combining Ant colony algorithm with Genetic algorithm and KM algorithm to find the overall service composition solutions with the maximum overall QoS utility for multiple users. However, these approaches focus on service selection for only one service at a time. Some researches focus on the service selection problem for multi-users considering their individual preferences [29-31]. In this problem, the requests of multi-users may be conflicted or correlated, which require more complicate coordinate effort to find the optimal services. Facing the challenge of selecting IoT service compositions for concurrent requests, Sun et al. reduced the problem to a constrained multi-objective optimization problem and took advantage of the PSO and GWO algorithm to address it [31]. Liu et al. proposed a service-oriented artificial bee colony algorithm to select service for concurrent requests [33]. The object of this method was to produce an optimal global service schema that improves the revenue of services providers and satisfy the requirement of requesters. Lima et al. [34] constructed a novel QoS model that classified the QoS attributes into three categories: exact, required and optimized. Based on this model, a utility model was proposed to evaluate the similarity of the QoS requirement and the service QoS. Then for each of the simultaneously incoming service choreography requests, the best-fit services which were not overload are selected. Although these researches can satisfy the load capacity constraint of services, the load balance can not be guaranteed. Wang et al. presented an interval number-based method to describe the service requirement and the service selection algorithm for multi-user [35]. A hierarchical clustering method was used to partition the candidate services into several service sets. Then the service with the best similarity with the requirement and the lowest load was chosen. In paper [36], Xiao et al. modelled the problem of cloud service composition of competing multi-users as a non-cooperative game and presented an iterative proximate algorithm to find the Nash equilibrium solution. Although the above studies tried to provide an equivalent opportunity for service providers, they did not consider the dynamic nature of service QoS, which is an important feature to be investigated to realized service coordination

and service load balance.

## 4. Problem Definition

In this section, we formalize the problem of service composition considering the dynamic QoS adjustment and load balance. Firstly, the preliminary definitions of related concepts are presented, which include web service, QoS, abstract workflow, composite service, service request and request queue. Then, the problem definition is given formally.

### 4.1 Preliminary Definition of the Related Concept

**Definition 1. (Web Service).** A web service is a 5-tuple $s = \{id, t, f, QoS, load\}$, where:
- *id*: is the identifier of the service;
- *t*: is the detail technical information of the service, including the endpoint, input and output parameters;
- *f*: is the functional description of the service, serving as the advertisement to support service discovery;
- *QoS*: is a set of non-functional attributes of the service, including response time, reliability, reputation and cost, etc;
- *load*: is the workload of the service; it is determined by the number of requests to the service in a given period of time.

**Definition 2. (QoS).** QoS is a set of non-functional attributes, $QoS = \{q_1, q_2, \dots, q_n\}$, and each attribute is a 3-tuple $q_i = \{type, v_d, p\}$, where:
- type: is the type of the QoS attribute;
- $v_d$: is the default value of the attribute;
- p: is the maximum percentage that the QoS attribute can change, which ranges from -1 to 1;

The definition of QoS indicates that the actual value of $q_i$ is between $v_d$ and $v_d \times (1 - p)$. The actual value of $q_i$ can be decided by the service owner or service running environment. For example, the response time could increase as the workload of service raised; the price of the service execution can be cut due to the service owner want to attract more clients. If p is zero, then the value of the attribute will remain unchanged. At this point, the QoS attributes can be classified into two categories: changeable and unchangeable.

Note that the QoS attributes can be positive or negative, where the higher the QoS value, the better the quality, or the higher the QoS value, the worse the quality. For instance, the attribute reliability is a positive attribute, while the attribute cost is a negative attribute. So for the positive attribute, if $q_i.p < 0$, it means that actual QoS value would be better than default value; in contrast, if $q_i.p > 0$, the actual QoS value would be worse than the default value. For the same reason, for the negative attribute, if $q_i.p < 0$, it means that actual QoS value would be worse; if $q_i.p > 0$, the actual QoS value would be better.

**Definition 3. (abstract workflow).** An abstract workflow is a 2-tuple $aw = \{VS, st\}$, where:
- VS: is a set of virtual service $VS = \{vs_1, vs_2, \dots, vs_m\}$, where $vs_i$ describes the functional requirement of the services that would be included in the workflow;
- *st*: is the structure of the abstract workflow, which defines the relationships between virtual services. The relationship can be a sequence, parallel, loop and alternative. In this paper, for simplicity, only the sequential structure is considered;

The abstract workflow cannot be invoked because it only comprises virtual services, as the workflow presented in **Fig. 1**. After all of the virtual services are replaced with concrete services, the abstract workflow will turn to be executable workflow.

**Definition 4. (composite service).** Composite service is a 3-tuple $cs = \{S, st, Q\}$, where:
- S: is a set of concrete services, which are defined in definition 1;
- st: is the structure of composite service, which is defined in definition 3;
- Q: is the overall QoS of the WS;

Composite service can be viewed as an instance of abstract workflow where all virtual services are replaced with concrete services. The overall QoS of composite service is determined by the QoS of component services and the structure of composite service. There are usually two ways to calculate the overall QoS, one is Pareto based method, the other one is the weighted average method [37, 38]. In this paper, we utilized the weighted average method as described in our previous work [39].

**Definition 5. (service request).** Service request is a 3-tuple $r = \{id, time, QC\}$, where:
- id: is the identifier of the request;
- time: is the timestamp when the request arrives at the server;
- QC: is the QoS constraint of the request;

The service request is sent by the user to invoke a composite service. The QC proposes the restriction on what kind of concrete component services that can be included in the composite service. The timestamp of request makes sure that the request can be processed in proper order.

**Definition 6. (request queue).** Request queue is the list of requests $RQ = \{r_1, r_2, \dots, r_l\}$. By pushing the new coming request into the request queue, we can make sure that the first incoming request will be served first.

### 4.2 Problem Statement

Based on the above definition, we formally present the problem of dynamic QoS adjustment enabled and load-balancing-aware service composition for multiple requests. Given an abstract workflow as and a set of web services $s = \{s_1, s_2, \dots, s_n\}$, for a request queue $RQ = \{r_1, r_2, \dots, r_l\}$, to find a composite service $cs_i$ for each service request $r_i$ in the request queue, and to optimize the following goals:

$$\min \frac{\sum_{i=0}^{l-1} u(cs_i.Q)}{l} \tag{1}$$

$$\min \frac{SD(s.load)}{\sum_{i=0}^{n-1} s_i.load/n} \tag{2}$$

$$\max \frac{\sum_{i=0}^{l-1} success(r_i)}{l} \tag{3}$$

$$success(r_i) = \begin{cases} 1, if\ cs_i.Q \succcurlyeq r_i.QC \\ 0, if\ cs_i.Q \prec r_i.QC \end{cases} \tag{4}$$

where $u(cs_i.Q)$ is the utility value of the QoS of the ith composite service. As mentioned in Definition 4, the utility value is calculated by the weighted average method as described in the paper [28]. In this paper, we assume that the lower of the utility, the better of quality. Hence the formula (1) tries to minimize the average utility of all generated composite services. $SD(s.load)$ is the standard deviation of the workload of web services, so formula (2) aims to keep load balance between services. In formula (4), $cs_i.Q \succcurlyeq r_i.QC$ means the QoS of $cs_i$ does

not exceed the QoS constraint of $r_i$, in contrast, $cs_i.Q \prec r_i.QC$ indicates that $cs_i$ does not satisfy the QoS requirement of $r_i$. In formula (3), the numerator stands for the number of successfully fulfilled requests. Then formula (3) means the fulfilled rate of requests shall be as high as possible.

From the problem statement, it can be concluded that the problem to be studied in this paper is a typical MOOP problem (multiple-objective optimal problem). As the other MOOP problems, the optimization goals may be conflict with others. For instance, the goals presented by formula (1) and (2) are the pair of conflicting goals. As we pursue the better quality of composite services, the more likely that the services in minority with better QoS will be chosen which leads to the unbalance between services. When the dynamic characteristic of the QoS attribute is considered, it is more difficult to tackle this problem efficiently.

## 5. DCBC Approach

In this section, we describe the DCBC approach in detail. The main idea and the framework of the DCBC approach are explained at first. Then the services preprocessing algorithm is presented. Finally, we briefly describe the algorithm to select candidate web services.

### 5.1 The Main Idea and the Framework of the DCBC Approach

**Fig. 2** demonstrates the framework of the service composition for multiple requests. In which, there are three parts: 1) the users who launch the service requests; 2) the service broker who receives the requests and responses for generating the service composition solutions; 3) the web services provided by service owners. In a given period of time, the requests arrive at the broker randomly. Similar to the other many online systems, the requests are pushed into the request queue in a FIFO manner to wait for being processed. Then the broker will remove the first request in the head of the queue every time.

According to the framework, to improve the throughput, the requests are arranged in a FIFO queue, which means the broker can process only one request at a time. It is different from the other researches in [26] that there are a group of requests that can be processed simultaneously. In this paper, for a request, we need to find a composite service which has better QoS and does not violate the QoS constraint according to the current work-load of each candidate services. The straightforward way to generate a solution is to compare all of the possible combinations of the concrete services, which will lead to the explosion of solution space. It is not feasible in real application especially the situation that there are a lot of concrete services for each virtual service of the abstract workflow.

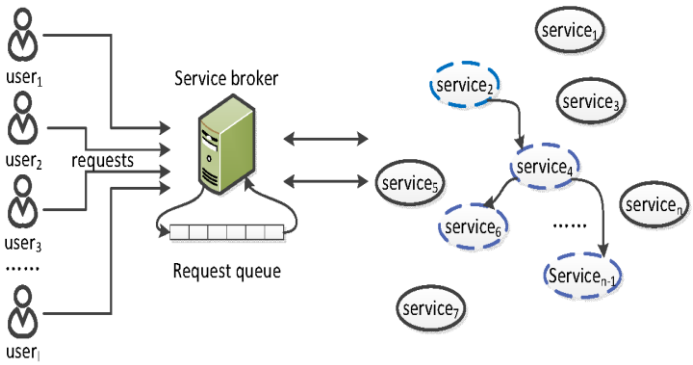


**Fig. 2.** The Framework of the Service Composition for Multiple Requests

To avoid the exhausting search for all combinations, we select an appropriate concrete service for each of virtual service. The key issue is how to ensure the chosen concrete service is the suitable component service of the feasible composite service, and if the QoS attributes of the chosen service are changeable, how can we determine the actual QoS value. To tackle this problem, the DCBC approach takes two steps, which are services preprocessing step and service selection step. The details of the two steps are described in section 5.2 and section 5.3 respectively.

## 5.2 Preprocessing of Concrete Web Services

The purpose of the preprocessing step is to generate information to filter out unsuitable concrete services rapidly. For a request $r$, we need to go through all of the virtual services $\{vs_1, vs_2, \ldots, vs_m\}$ one by one and select a concrete service for each of them. Supposing that the service $s_{ij}$ is a concrete service for the virtual service $vs_i$, to verify if $s_{ij}$ can be selected as the candidate component of the feasible composite service, we can simply calculate the overall utility value of $s_{ij}$ with the services that have been chosen and the best concrete services in the remain virtual services, as the **Fig. 3** demonstrates. If the overall utility value satisfies the constraint of r, then the service $s_{ij}$ can be reserved for further selection, otherwise, $s_{ij}$ must be not the candidate service to fulfill $r$..

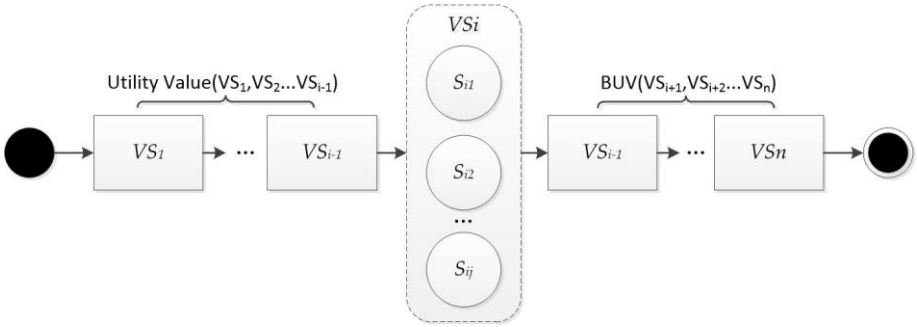


**Fig. 3.** Example of verifying the feasibility of concrete services $s_{ij}$

To facilitate the process, for each virtual service $vs_i$, we calculate and store the best overall QoS utility value (BUV) of $vs_i's$ successive services. As the QoS attribute may be changeable or unchangeable, BUV also can be classified into two folders, which are fixed one and dynamic one respectively. The fixed BUV is calculated according to the original QoS value, while the dynamic BUV is calculated with the consideration of the QoS fluctuation. The following formulas (5) and (6) are employed to verify the feasibility of concrete service $s_{ij}$

$$u(VS_1, VS_2 \ldots VS_n) = u(VS_1, VS_2 \ldots VS_{i-1}) + u(VS_{ij}) + BUV(VS_{i+1}, VS_{i+2} \ldots VS_n) \quad (5)$$

$$feasible(VS_{ij}) = \begin{cases} 1, if\ u(VS_1, VS_2 \ldots VS_n) \succcurlyeq r.QC \\ 0, u(VS_1, VS_2 \ldots VS_n) \prec r.QC \end{cases} \quad (6)$$

The algorithm 1 shows the detail of the concrete web services preprocessing step. The algorithm takes an abstract workflow and a set of concrete web services as input and produces a set of BUVs. The algorithm mainly contains two steps. The first step, Line 2-7, calculates the fixed BUV and dynamic BUV of each virtual service. The function getServices() is to retrieve the concrete services which satisfy the functional requirement of virtual service. Then

the function getBestFixedQoS() and getBestDynQoS() at line 4 and 5 gain the fixed BUV and dynamic BUV respectively by comparing services one by one. The second step, from line 8 to 11, performs a backwards BUV calculation. We accumulate the fixed BUV and dynamic BUV of the latter virtual service to its previous one in each loop. So after the loop, the BUV(vs1) will store the best fixed and dynamic overall QoS utility value of the abstract workflow $aw$, which are constituted by the optimal concrete web service of each virtual service.

**Algorithm 1:** Preprocessing the Concrete Web Services

**Input:** $aw = \{VS = \{vs_1, vs_2, \dots, vs_m\}, st\}$, $set(s) = \{s_1, s_2 \dots s_n\}$
**Output:** a set of BUV

1. *set*(BUV)=∅
2. **for** *i*=1 **to** *m* **do**
3. *set*($vs_i$)=getServices(*set*(*s*), $vs_i$)
4. buv($vs_i$).FQoS=getBestFixedQoS (*set*($vs_i$))
5. buv ($vs_i$).DynQoS=getBestDynQoS (*set*($vs_i$))
6. *set*(BUV).add(buv($vs_i$))
7. **end for**
8. **for** *i*=*m*-1 **to** 1 **do**
9. buv($vs_i$).FQoS += buv($vs_{i+1}$).FQoS
10. buv($vs_i$).DynQoS += buv($vs_{i+1}$).DynQoS
11. **end for**
12. **return** *set*(BUV)

Taking the **Fig. 1** and **Table 1** as an example, **Table 3** presents the procedure and result of the preprocessing for the workflow which depicted in **Fig. 1**. *BUV.FQoS* represents the fixed BUV and *BUV.DynQoS* stands for the dynamical BUV.As the direction of arrows shows, step 1 starts from the first virtual service $vs_1$ to the final virtual service $vs_3$. Since that the price attribute is the only attribute considered in this example, so the BUV is calculated based on the lowest price. The step 2 starts from the final virtual service and goes backwards to the first one. In this way, the BUV of latter virtual service is accumulated to the previous one. Finally, a set of BUV is produced and ready for providing decision making information for service selection in the next step of the DCBC approach.

**Table 3.** The Preprocessing Example

| | $vs_1$ | $vs_2$ | $vs_3$ |
|---|---|---|---|
| | *Setp 1:* → | | |
| *BUV.FQoS* | 80 | 30 | 27 |
| *BUV.DynQoS* | 72 | 30 | 24 |
| | *Setp 2:* ← | | |
| *BUV.FQoS* | 137 | 57 | 27 |
| *BUV.DynQoS* | 126 | 54 | 24 |

The time complexity of the preprocessing algorithm is $O(m \times p)$, where m is the number of virtual services and p is the average number of concrete web services of the virtual services. Obviously, the time complexity of the preprocessing algorithm is low and would not impose

extra load on the service composition.

### 5.3 Service Selection with the Consideration of Load Balance and Dynamic QoS

In the online application scenarios, the response time is a critical indicator. Especially in the scenario that there are lots of requests flow in a short time. Therefore, it is important to acquire a feasible composite service for the request ASAP. To achieve this goal, the algorithm 2 tries to select concrete service by using a dynamic programming like idea, which selects concrete service for a virtual service only according to the current state in each step. The following pseudo code describes the logic of service selection.

**Algorithm 2: Service Selection for Requests**

**Input:** $aw = \{VS = \{vs_1, vs_2, \dots, vs_m\}, st\}$,
$set(s) = \{s_1, s_2 \dots s_n\}$, $RQ = \{r_1, r_2, \dots, r_l\}$,
*set*(BUV)= $\{\text{buv}(vs_1), \text{buv}(vs_2), \dots, \text{buv}(vs_m)\}$,
**Output:** a set of composite services

1. *set*(CS)=∅
2. **for** *i*=1 **to** *l* **do**
3. **if** buv($vs_1$).FQoS ≺ $r_i$.QC **and** buv($vs_1$).DynQoS ≺ $r_i$.QC **then**
4. continue;
5. **end if**
6. $cs_i$=∅
7. $cs_i$.Q=0
8. **for** j=1 **to** m **do**
9. *set*($vs_j$)=getServices(*set*(*s*), $vs_j$)
10. **if j<m do**
11. restBUV=better**(**buv$\left(vs_j\right)$. FQoS, buv$\left(vs_j\right)$. FQoS**)**
12. **else**
13. restBUV=0
14. **end if**
15. **for each** s **in** *set*($vs_j$) **do**
16. avgLoad=calAvgLoad(*set*($vs_j$))
17. **if** (s.load<avgLoad **or** (s.QoS+restBUV+$cs_i$.Q) ≺ $r_i$.QC) **then**
18. s.QoS=s.QoS× (1 − p)
19. **end if**
20. **if** ((s.QoS+restBUV+$cs_i$.Q) ≽ $r_i$.QC) **then**
21. s.probability=**calProbability**(s)
22. **else**
23. continue
24. **end if**
25. **end for**
26. service=selectServiceWithMaxPro(*set*($vs_j$))
27. $cs_i$.add(service)
28. $cs_i$.Q+=service.QoS

29. **end for**
30. *set*(CS).add($cs_i$)
31. **end for**
32. **return** *set*(CS)

In the beginning, the set of composite services is initialized to be empty at line 1. The composite services are the solutions for users' requests. The main operations of the algorithm 2 are contained in a loop structure, which starts from line 2 to line 31. In one cycle, the algorithm would generate a composite service for one of the requests. Firstly, the algorithm checks if the request can be fulfilled at line 3. Benefit from the algorithm 1, we can do this check quickly by comparing the BUV($vs_1$) with the QoS constraint of the request $r_i$. If the best composite service can not meet the requirement of the request $r_i$, then $r_i$ will be skipped. Otherwise, there must be a feasible solution for $r_i$. Secondly, from line 8 to line 30, it selects concrete service for each of the virtual services. For a virtual service, the algorithm goes over all of the concrete services that belong to it and calculates their selection probabilities. The selection probability indicates the chance of concrete service to be chosen as a component service for $r_i$. The line 21 calculates the selection probability for a service according to the following formulation:

$$p_{ij} = \frac{A}{u(s_{ij}.QoS)\times(1+s_{ij}.load)^{\alpha}} \tag{7}$$

Where $p_{ij}$ denotes the selection probability of the jth concrete service of the ith virtual service; $u(s_{ij}.QoS)$ returns the utility value of $s_{ij}$'s QoS, here we assume that the better of the QoS, the smaller the utility value; α means the importance of the load of service, which is not less than zero; A is a const positive integer. From the formula (7), we can see that the selection probability is decided by the QoS and the workload of service. The services with better QoS and lower workload are more likely to be selected. Remind that the QoS of service may not be fixed, in lines 17-19, the service can change its QoS according to the following criteria: 1) if the fixed QoS of the service $s_{ij}$ can not meet the constraints of the request $r_i$; 2) if the workload of service $s_{ij}$ is lower than the average workload of concrete services that relate to $vs_i$. If one of the criteria is met, the service can change its QoS as line 18 does. The first criterion indicates that a service can improve the service quality dynamically to fulfill the request and win the service competition with the other concrete services. The second criterion makes sure that the workload of services can be kept balance. We can change the value of the parameter α to select service by emphasizing on the QoS or load balance. If the parameter α is set to be zero, then the service selection strategy is equivalent to QoS-aware selection without considering load balance, which will result to the situation that only the services with best QoS can be composed for every request. Due to the consideration for balancing QoS and workload, the value of parameter α must be set greater than zero. In section 6, we carry out an experiment to evaluate the impact of parameter α on the quality of composition solutions. Line 20 checks if the current concrete service $s_{ij}$ can meet the constraint of request. According to the formula (6), if the sum of the QoS of $s_{ij}$, the BUV of reset of services and the current QoS of selected service can not meet the constraint, then the service $s_{ij}$ will be skipped. Finally, in line 26-27, all of the qualified candidate services will be compared according to the selection probability, then the Roulette Wheel selection mechanism is applied to choose the component service of $cs_i$ from the top 20% candidate services according to the probability.

The time complexity of the algorithm 2 is $O(l \times m \times p)$, where l is the number of requests, m is the number of virtual services and p is the average number of concrete web services of the virtual services. Combined with the algorithm 1, the time complexity of the DCBC method is $O(l \times m \times p)$. On the basis of the service preprocessing, the running time of the DCBC method is in a low order of magnitude.

## 6. Experiments

To evaluate the effectiveness of the DCBC method, we carry out several experiments in this section. The first set of experiments is to compare the effectiveness of the DCBC approach with other methods. The second set of experiments is to evaluate the impact of parameters on the DCBC method. These experiments mainly focus on the quality of solutions, the load balance of services and the number of requests being fulfilled, which are measured by the formulas (1), (2) and (3) that presented in section 4.2.

We generate a set of synthetic data for the experiments. The experiment dataset includes information about abstract workflows, concrete services and service requests. For each of concrete service, the price and response time attributes are concerned. The major parameters to generate the experiment dataset are the number of virtual services, the number of concrete services for each virtual service, the percent of concrete services that has dynamic QoS, and the number of requests. The detail information of the experiment data will be described in the following sections.

The related algorithms are implemented in C++ by visual studio 2017. All of the experiments are performed on a laptop with an Intel I7-7500 CPU, 8 GB RAM and Windows 10 operating system.

### 6.1 Effectiveness Evaluation of DCBC Approach

In this section, we compare the DCBC approach with the other three methods, which are the BoC, DoC and QDEC method respectively. The BoC method is the Balance-only-Composition version of DCBC which tries to keep the balance of services while without considering the dynamic QoS. Specifically, compared to the DCBC method, the BoC method lacks of the line16 to line 19 in algorithm 2. In contrast, the DoC method is the Dynamic-only-Composition version which supports the adjustment of QoS value in runtime, but the load balance is not guaranteed. In DoC method, the mechanism of QoS adjustment is different with the DCBC approach in that the Qos adjustment occurs only in the case that the QoS of service cannot satisfy with the requirement of requests. Therefore, the line 17 of algorithm 2 of DoC method shall be: *if* $((s.QoS+restBUV+cs_i.Q) \prec r_i.QC)$. Furthermore, the parameter $\alpha$ of formula (7) in DoC method is set to be zero. The QDEC method is proposed in paper [37], which adopts global QoS decomposition and Kuhn-Munkres algorithm to select services for each request.

In this evaluation, the number of tasks in the workflow is set to be 30; the number of concrete services for each task ranges from 50 to 60. In the DCBC approach, 10 percent of services are randomly chosen to support the dynamical QoS changing. And then the changeable space of the QoS attribute is randomly set to between 0 and 10%. The number of service requests ranges from 100 to 1000.

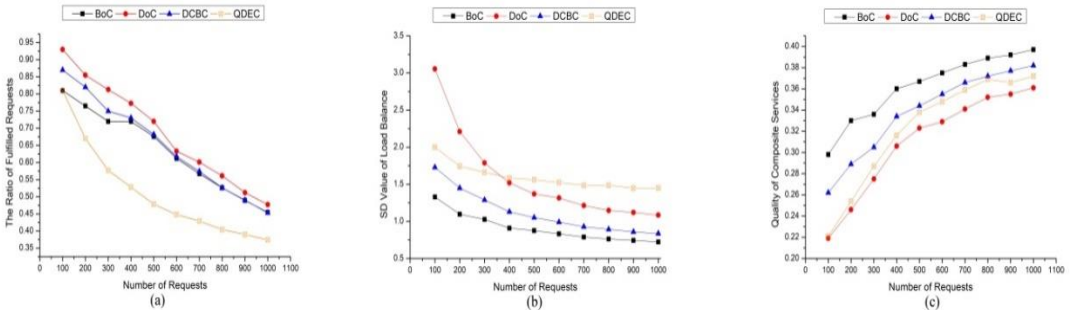


**Fig. 4.** Effectiveness Comparison

**Fig. 4** shows the experiment result. The part (a) demonstrates the percent of successful requests to the total service requests. It can be seen that the rate of fulfilled requests of the four methods decreases as the number of requests increases, which is because that there are fewer services available for the later requests as the number of requests increases. But the DCBC and DoC algorithm is always better than the other two. The reason is that the DCBC and DoC algorithm try to meet the requirement of requests as far as possible by adjusting the QoS of service. From the part (b), we can see that the DCBC and BoC algorithm can maintain better load-balance than DoC and QDEC. This is due to that the DoC algorithm always selects the services with better QoS, which leads to the result that some other services have no chance to be invoked. The part (c) demonstrates the quality of composition services produced by the four algorithms, where the vertical coordinate shows the utility value of composition services. The figure indicates that the DoC and QDEC algorithm can generate solutions with better quality. The above experiments reveal that the DCBC method achieves a balance between load-balance and solution quality. The BoC and DoC algorithm can be seen as the extreme versions of the DCBC method which emphasizes on load-balance and quality respectively. If only the load-balance is considered, the request satisfaction rate will decrease, as the part (1) shows. If only the dynamic QoS is considered, the quality of solutions can not be guaranteed, as the part (2) shows. So the DCBC method achieves better request satisfaction rate and load balancing by losing the quality of solutions.

### 6.2 Analysis of Key Parameters

In this section, a set of experiments are carried out to analyze the impact of key parameters on the efficiency of the DCBC method. These parameters include the importance factor of load-balance α, the number of virtual services, the number of concrete services, and the percent of services which support dynamical QoS changing.

**Fig. 5** shows the result of the evaluation of the important factor of load-balance α. To test the impact of α, we set the value of α from 0 to 2 and step by 0.2. The number of requests is set to be 1000, the number of virtual services is 30, and the number of concrete services for each virtual service is between 190 and 200. The part (a) of **Fig. 5** demonstrates how the request satisfaction rate is affected by α. As the value of α increases, the request satisfaction rate drops gradually, since some candidate services with better QoS may not be chosen to keep balance between services. This is the same reason to cause the decline of QoS, as the part (c) of **Fig. 5** shows. The part (b) illustrates that the services reach better load balance as the value of α grows. From **Fig. 5**, it can be concluded that the value of α is not the larger the better, we need to weigh and consider the load-balance and service quality.

**Fig. 6** shows the impact of various numbers of virtual services. The number of virtual services indicates the complexity of the workflow. In this experiment, the number of requests

is set to be 1000, the number of concrete services for each virtual service is between 190 and 200, and the number of virtual services ranges from 5 to 50 and step by 5. From the part (a), we can see that the request satisfaction ratio stays steady when the number of virtual services goes from 5 to 25, but it decreases as the number of virtual services goes from 30 to 50. The same phenomenon can be observed from part (b). It indicates that the number of virtual services has negative impact on the effect of the DCBC method. But the quality of composite services can keep stable as the number of virtual services grows, which is demonstrated in part (c).

**Fig. 7** shows the impact of various numbers of concrete services. In this experiment, the number of requests is 1000, the number of virtual services is 30, and the number of concrete services ranges from 20 to 200 and step by 20. The part (a) shows that the request satisfaction rate increases from about 27% to near 80% when the number of concrete services goes from 20 to 200. It is because that the more candidate services provide the more chances to fulfill user's requests. Conversely, it is difficult to maintain load-balance as the number of concrete services increases, as which revealed by the part (b). The part (c) shows that the quality of composite services improves with the number of concrete services raises.

**Fig. 8** demonstrates how the percent of services with dynamic QoS affect the effectiveness of the DCBC method. In this experiment, the number of requests is set to 1000, the number of concrete services for each virtual service is between 190 and 200, and the percent of services with dynamic QoS ranges from 10% to 50% and step by 10%. The part (a) shows that the request satisfaction rate increases as the percent of services with dynamic QoS increases. The more services could adjust the QoS, the more requests could be fulfilled. In part (b), we can see that it is more unbalance between services when the number of services with dynamic QoS is less, but it finally reaches better balance when the number of services with dynamic QoS increased. The part (c) also reveals that the quality of composition services improves as the percent of services with dynamic QoS increases. This experiment indicates that the dynamic QoS is critical to producing the better composition services, and it could not be ignored.

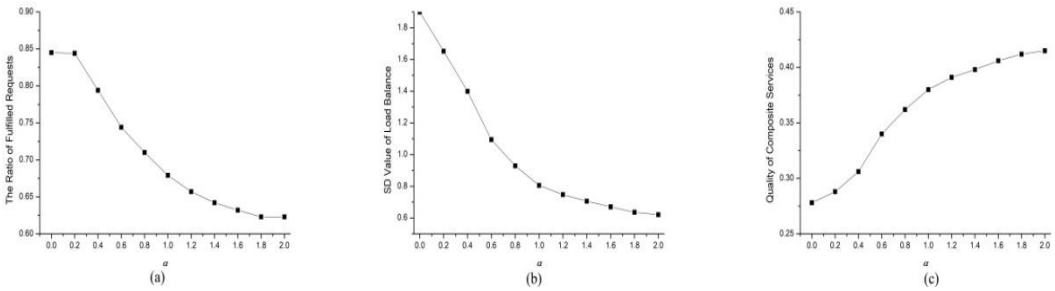


**Fig. 5.** Evaluation of the impact of the importance factor

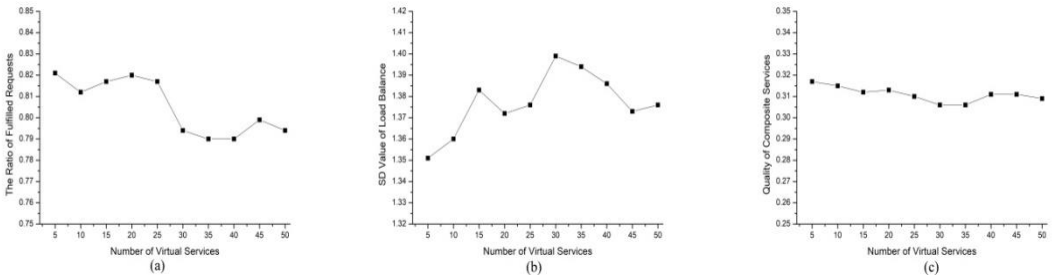


**Fig. 6.** Evaluation of the impact of the number of virtual services

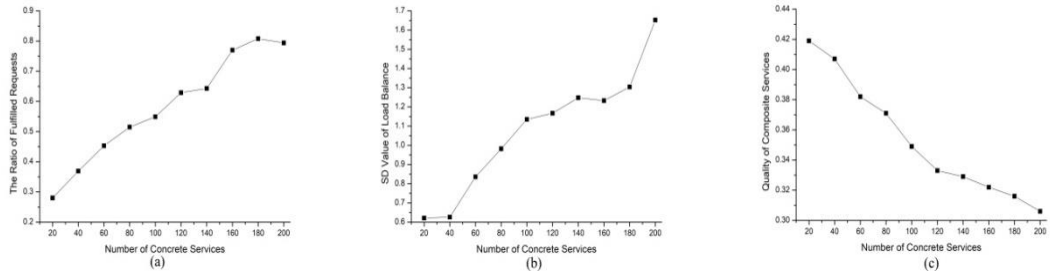


**Fig. 7.** Evaluation of the impact of the number of concrete services

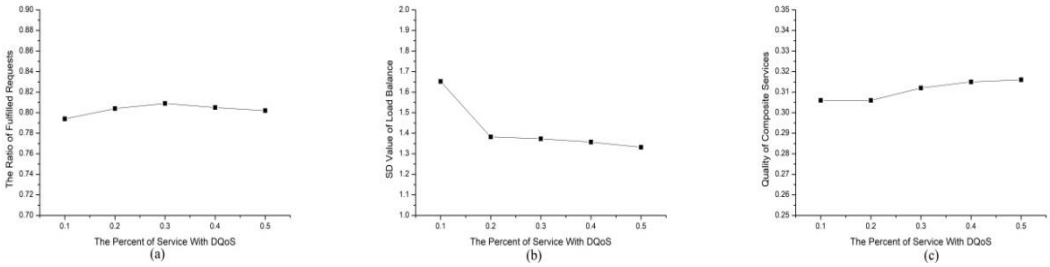


**Fig. 8.** Evaluation of the impact of the percent of service with DQoS

## 7. Conclusion

Service selection with optimal overall QoS and keep the load balance between services is an important issue in the field of service composition. In this paper, a novel method, DCBC, is proposed to tackle this problem. In this method, we assume that the QoS of service is not static and can be changed to a certain extent. The DCBC method encourages service providers to dynamically improve the service QoS to gain more requesters. To achieve the goal of load balance, the service with a lower load will provide better QoS to be chosen. The two steps of the DCBC method, web services preprocessing and service selection, are introduced to facilitate the process of service composition. The experiments show that the DCBC method can generate the optimal composite service effectively.

However, the problem of requests over satisfaction still exists in our proposed method, especially in the initiating stage. For this reason, in the next research, we will extend our work to provide composite service with proper QoS according to the user's request. Furthermore, the QoS adjustment method is too rigid in the current work, which is another concern in the next research.

## References

[1] L. Zhang, Y. Luo, F. Tao, B. H. Le, L. Ren, X. Zhang, H. Guo, Y. Cheng, A. Hu, and Y. Liu, "Cloud manufacturing: a new manufacturing paradigm," *Enterprise Information Systems*, vol. 8, no. 2, pp. 167-187, 2014. Article (CrossRef Link)

[2] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, and C. Wu, "Category-aware API clustering and distributed recommendation for automatic mashup creation," *IEEE Transactions on Services Computing*, vol. 8, no. 5, pp. 674-687, 2014. Article (CrossRef Link)

[3] C. Jatoth, G. R. Gangadharan, and R. Buyya, "Computational intelligence based QoS-aware web service composition: A systematic literature review," *IEEE Transactions on Services Computing*, vol. 10, no. 3, pp. 475-492, 2015. Article (CrossRef Link)
[4] H. Zheng, J. Yang, and W. Zhao, "Probabilistic QoS aggregations for service composition," *ACM Transactions on the Web (TWEB)*, vol. 10, no. 2, 2016. Article (CrossRef Link)
[5] A. Naseri and N. J. Navimipour, "A new agent-based method for QoS-aware cloud service composition using particle swarm optimization algorithm," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 1851-1864, 2019. Article (CrossRef Link)
[6] B. Bhattu, C. Jatoth, G. R. Gangadharan, and U. Fiore, "A MapReduce-based modified Grey Wolf optimizer for QoS-aware big service composition," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 8, 2020. Article (CrossRef Link)
[7] S. Niu, G. Zou, Y. Gan, Y. Xiang, and B. Zhang, "Towards the optimality of QoS-aware web service composition with uncertainty," *International Journal of Web and Grid Services*, vol. 15, no. 1, 2019. Article (CrossRef Link)
[8] X. Wang, X. Xu, Q. Z. Sheng, Z. Wang, and L. Yao, "Novel Artificial Bee Colony Algorithms for QoS-Aware Service Selection," *IEEE Transactions* on *Services Computing*, vol. 12, no. 2, pp. 247-261, 2019. Article (CrossRef Link)
[9] S. Bhushan and P. C. H. Reddy, "A hybrid meta-heuristic approach for QoS-aware cloud service composition," *International Journal of Web Services Research (IJWSR)*, vol. 15, no. 2, pp. 1-20, 2018. Article (CrossRef Link)
[10] C. Li, J. Li and H. Chen, "A Meta-Heuristic-Based Approach for Qos-Aware Service Composition," *IEEE Access*, vol. 8, pp. 69579-69592, 2020. Article (CrossRef Link)
[11] H. Fekih, S. Mtibaa, and S. Bouamama, "An Efficient User-Centric Web Service Composition Based on Harmony Particle Swarm Optimization," *International Journal of Web Services Research*, vol. 16, no. 1, pp. 1-21, 2019. Article (CrossRef Link)
[12] J. Lartigau, X. Xu, L. Nie, and D. Zhan, "Cloud manufacturing service composition based on QoS with geo-perspective transportation using an improved Artificial Bee Colony optimisation algorithm," *International Journal of Production Research*, vol. 53, no. 14, pp. 4380-4404, 2015. Article (CrossRef Link)
[13] E. M. Khanouche, H. Gadouche, Z. Farah, and A. Tari, "Flexible QoS-aware services composition for Service Computing environments," *Computer Networks*, vol. 166, 2019. Article (CrossRef Link)
[14] S. K. Gavvala, C. Jatoth, G. R. Gangadharan, and R. Buyya, "QpS-aware cloud service composition using eagle strategy," *Future Generation Computer Systems*, vol. 90, pp. 273-290, 2019. Article (CrossRef Link)
[15] X. Dong, Q. Liu, D. Lu, S. Ma, and J. Zheng, "QoS-Aware Path Finding and Load Balancing in Service-Composition," in *Proc. of International Conference on Networking and Network Applications*, pp. 409-414, 2019. Article (CrossRef Link)
[16] C. Lv, W. Jiang, S. Hu, J. Wang, G. Lu, and Z. Liu, "Efficient dynamic evolution of service composition," *IEEE Transactions on Services Computing*, vol. 11, no. 4, pp. 630-643, 2015. Article (CrossRef Link)
[17] L. Barakat, S. Miles, and M. Luck, "Adaptive composition in dynamic service environments," *Future Generations Computer System*, vol. 80, pp. 215-228, 2019. Article (CrossRef Link)
[18] Y. Que, W. Zhong, H. Chen, X. Chen, and X. Ji, "Improved adaptive immune genetic algorithm for optimal QoS-aware service composition selection in cloud manufacturing," *The International Journal of Advanced Manufacturing Technology*, vol. 96, pp. 4455-4465, 2018. Article (CrossRef Link)
[19] P. Wang, J. Meng, T. Liu, Y. Zhan, W. T. Tsai, and Z. Jin, "Smart-Contract based Negotiation for Adaptive QoS-aware Service Composition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1403-1420, 2018. Article (CrossRef Link)
[20] E. Khanfir, R. B. Djmeaa, and I. Amous, "Self-Adaptive Goal-Driven Web Service Composition Based on Context and QoS," in *Proc. of IEEE 14th International Conference on e-Business Engineering*, pp. 201-207, 2017. Article (CrossRef Link)

[21] S. L. Chen, Y. Y. Chen, and S. H. Kuo, "CLB: A novel load balancing architecture and algorithm for cloud services," *Computers & Electrical Engineering*, vol. 58, pp. 154-160, 2017. Article (CrossRef Link)
[22] O. Hioual, Z. Boufaïda, and S. M. Hemam, "Load balancing, cost and response time minimisation issues in agent-based multi cloud service composition," *International Journal of Internet Protocol Technology*, vol. 2, no. 2, 2017. Article (CrossRef Link)
[23] X. Xiao, Y. M. Kou, S. F. Wang, and Z. Z. Liu, "Computational experiment research on the equalization-oriented service strategy in collaborative manufacturing," *IEEE Transactions on Services Computing*, vol. 11, no. 2, pp. 369-383, 2016. Article (CrossRef Link)
[24] X. Wang, J. Liu, B. Cao, and M. Tang, "A global optimal service selection approach based on QoS and load-aware in cloud environment," in *Proc. of 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE Inter-national Conference on Embedded and Ubiquitous Computing*, pp. 762-768, 2013. Article (CrossRef Link)
[25] G. Kang, J. Liu, M. Tang, X. Liu, and K. K. Fletcher, "Web service selection for resolving conflicting service requests," in *Proc. of IEEE International Conference on Web Services*, pp. 387-394, 2011. Article (CrossRef Link)
[26] A. S. Kurdija, M. Šilić, G. Delač, K. Vladimir, and S. Srbljić, "Efficient Multi-user Service Selection Based on the Transportation Problem," in *Proc. of International Conference on Web Services*, pp. 507-515, 2018. Article (CrossRef Link)
[27] S. Wang, C. H. Hsu, Z. Liang, Q. Sun, and F. Yang, "Multi-user web service selection based on multi-QoS prediction," *Information Systems Frontier*, vol. 16, pp. 143-152, 2014. Article (CrossRef Link)
[28] H. Jin, H. Zou, F. Yang, R. Lin, and X. Zhao, "A hybrid service selection approach for multi-user requests," in *Proc. of 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pp. 1142-1149, 2012. Article (CrossRef Link)
[29] B. Heinrich, L. Lewerenz, M. Mayer, and M. Klier, "Enhancing Decision Support in Multi User Service Selection," in *Proc. of International Conference on Information Systems*, pp. 1-20, 2015. Article (CrossRef Link)
[30] M. Mayer, "Multi-user service re-selection: react dynamically to events occurring at process execution," in *Proc. of the 25th European Conference on Information Systems (ECIS)*, pp. 1807-1821, 2017. Article (CrossRef Link)
[31] B. Heinrich and M. Mayer, "Service selection in mobile environments: considering multiple users and context-awareness," *Journal of Decision Systems*, vol. 27, no. 2, pp. 92-122, 2018. Article (CrossRef Link)
[32] M. Sun, Z. Zhou, J. Wang, C. Du, and W. Gaaloul, "Energy-Efficient IoT Service Composition for Concurrent Timed Applications," *Future Generation Computer Systems*, vol. 100, pp. 1017-1030, 2019. Article (CrossRef Link)
[33] Z. Liu and X. Xu, "S-ABC - A Service-Oriented Artificial Bee Colony Algorithm for Global Optimal Services Selection in Concurrent Requests Environment," in *Proc. of IEEE International Conference on Web Service*, pp. 503-509, 2014. Article (CrossRef Link)
[34] J. C. Lima, R. C. A. Rocha, and F. M. Costa, "An approach for qos-aware selection of shared services for multiple service choreographies," in *Proc. of IEEE Symposium on Service-Oriented System Engineering*, pp. 221-230, 2016. Article (CrossRef Link)
[35] H. Wang and Y. Cheng, "Interval Number Based Service Selection for Multi-users' Requirements," in *Proc. of IEEE International Conference on Web Services (ICWS)*, pp. 712-715, 2016. Article (CrossRef Link)
[36] Z. Xiao, Y. Guo, G. Liu, and J. Du, "Noncooperative Optimization of Multi-user Request Strategy in Cloud Service Composition Reservation," in *Proc. of International Conference on Algorithms and Architectures for Parallel Processing*, vol. 11334, pp. 138-152, 2018. Article (CrossRef Link)
[37] S. Bhushan and P. C. H. Reddy, "A Hybrid Meta-Heuristic Approach for QoS-Aware Cloud Service Composition," *International Journal of Web Services Research*, vol. 15, no. 2, pp. 1-20, 2018. Article (CrossRef Link)

[38] O. Alsaryrah, I. Mashal, and T. Y. Chung, "Bi-objective optimization for energy aware IoT service composition," *IEEE Access*, vol. 6, pp. 26809-26819, 2018. Article (CrossRef Link)
[39] X. Wu, C. Chen, and H. Huang, "A Data Volume Aware Ant Colony Optimization Approach for Geographical Knowledge Cloud Service Composition," *International Journal of Grid and Distributed Computing*, vol. 9, no. 6, pp. 103-116, 2016. Article (CrossRef Link)

**Xiaozhu Wu** received the MSc degree in computer science from Fuzhou University, China, in 2005, the PhD degrees in communication&information system from Fuzhou University, China, in 2014. He is currently a lecture in college of mathematics and computer science at Fuzhou University, China. His fields of research are focused on service computing, spatial data mining & geographical knowledge grid/cloud.