QoS and SLA Aware Web Service Composition in Cloud Environment

Dandan Wang[1], Hao Ding[1] , Yang Yang[1], Zhenqiang Mi[1], Li Liu[2] and Zenggang Xiong[3]
[1] School of Computer and Communication Engineering, University of Science and Technology Beijing
Beijing 100083, China
[e-mail: wdd_ustb@163.com, haoding8724@gmail.com, yyang@ustb.edu.cn, mizq@ustb.edu.cn]
[2] School of Automation, University of Science and Technology Beijing
Beijing 100083, China
[e-mail: liuli@ustb.edu.cn]
[3]School of Computer and Information Science, Hubei Engineering University
Xiaogan, Hubei 432000, China.
[e-mail: jkxxzg2003@163.com]
*Corresponding author: *Zhenqiang Mi*

Received March 1, 2016; revised May 21, 2016; accepted October 24, 2016; published December 31, 2016

Abstract

As a service-oriented paradigm, web service composition has obtained great attention from both academia and industry, especially in the area of cloud service. Nowadays more and more web services providing the same function but different in QoS are available in cloud, so an important mission of service composition strategy is to select the optimal composition solution according to QoS. Furthermore, the selected composition solution should satisfy the service level agreement (SLA) which defines users' request for the performance of composite service, such as price and response time. A composite service is feasible only if its QoS satisfies user's request. In order to obtain composite service with the optimal QoS and avoid SLA violations simultaneously, in this paper we first propose a QoS evaluation method which takes the SLA satisfaction into account. Then we design a service selection algorithm based on our QoS evaluation method. At last, we put forward a parallel running strategy for the proposed selection algorithm. The simulation results show that our approach outperforms existing approaches in terms of solutions' optimality and feasibility. Through our running strategy, the computation time can be reduced to a large extent.

Keywords: Web service; QoS; SLA; service composition; cloud computing

A preliminary version of this paper appeared in CCBD 2015, November 4-6, Shanghai, China. This version includes more detailed descriptions of the proposed algorithm, new proposed running strategy and extensive simulation studies. This work was supported by the National Science Foundation of China (Grant No. 61272432, 61370092, 61370132 and 61472033), Hubei Provincial Department of Education Outstanding Youth Scientific Innovation Team Support Foundation (T20410) and Fundamental Research Funds for the Central Universities (TW201502).

http://dx.doi.org/10.3837/tiis.2016.12.006 **ISSN : 1976-7277**

1. Introduction

Web services can be released, discovered and utilized following a set of standards such as SOAP, WSDL, and UDDI [1][2]. In addition, they can communicate with each other through standard protocols and XML based messages. Given the convenience and reusability, web services have become a main mode of cloud application. However, the function of single web service is usually too simple to satisfy the user's complex demand. One solution to solve this problem is to integrate independent services into value-added composite service.

The system architecture for service composition in cloud environment is shown in **Fig. 1**. The cloud architecture includes three layers: software layer, platform layer and infrastructure layer. A user sends composition requests to brokers for utilizing composite web services. The software layer includes brokers and web services. The brokers, which can be centralized or decentralized, manage all services that are offered to users by SaaS providers. Web services are registered to brokers by service providers in order to be discovered. The composition engine in platform layer communicates with the brokers to discover candidate services according to the user's request. Based on the discovered candidate services, composition engine generates an execution plan which satisfies user's requirements. The infrastructure layer controls the actual resource allocation in terms of the execution plan generated by platform layer.

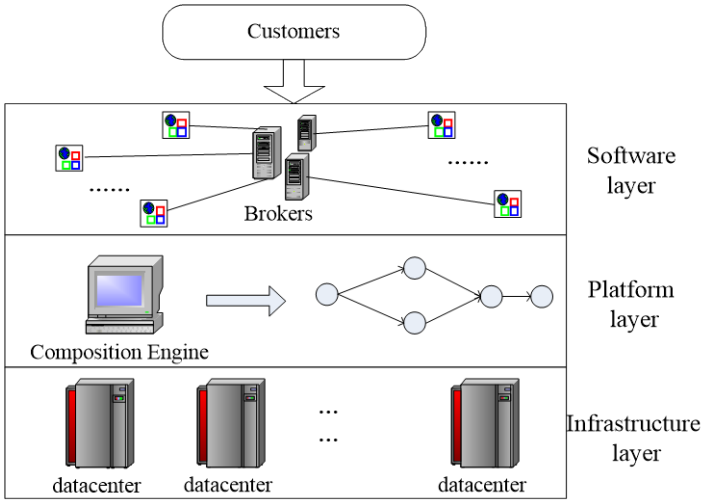


Fig. 1. System architecture for service composition in cloud environment

For example, in a government cloud for regional emergency response, a large number of web services providing different functions, such as geographic information, meteorological analysis, experts prognosis and materials distribution are deployed on software layer. When an emergency occurs, composition engine in platform layer need to compose these services to a complete workflow according to the predefined 'emergency plan'. The composed service must satisfy the function and performance demand defined in the 'emergency plan'. Because of the great significance for practice, service composition in cloud environment has attracted a lot of attention in both academia and industry.

Cloud environment and service composition are auxiliary to each other. On one hand, cloud environment provides enormous hardware and software resources which are readily accessible.

On the other hand, service composition provides an effective way for users to manage and utilize cloud resources. Given the feature of cloud environment, there are two challenges for web service composition in cloud environment.

First, with the number of cloud services growing, a number of services that provide the same function but differ in QoS are available. QoS of web services refers to various nonfunctional characteristics, such as response time and reliability. The composition approach needs to select the most appropriate service for each task so that the overall performance of the composite service is optimal. This is because high QoS can improve the reputation of service providers and then help to increase the providers' profit. However, QoS of composite service involves multiple attributes. So evaluation of the overall performance is a multiple dimensions computation problem and need to be adjusted to particular context.

Second, the QoS of composite service need to satisfy the constraints defined in service level agreement (SLA). SLA is a legal contract that states end-to-end QoS requirements between user and service provider [3]. The service provider must make compensation according to the agreement if the QoS of composite service violates SLA. Therefore, it is an important problem for service provider to avoid SLA violations.

To deal with these issues, we propose an approach towards QoS and SLA aware web service composition in cloud environment. We first give the process of service composition and model the composition problem as an optimization problem. Then we design a service evaluation method which transforms multiple QoS attributes of a composite service into a value. In our evaluation method, we consider the QoS constraints defined in SLA as important factors. Based on the evaluation method, we propose a service selection algorithm aiming at obtaining the optimal QoS and increasing solutions' feasibility. At last, we put forward a running strategy for the algorithm which can improve the solutions' optimality and reduce the computing time to a large extend. The main contribution of this paper can be stated as follows:

1. We map the problem of web service composition in cloud environment into a constrained composition optimization problem, and address it using a heuristic approach based on genetic and simulated annealing algorithm.

2. We propose a service evaluation method for detecting which composition scheme should be avoided because it may lead to SLA violations. The simulation results show that this method helps to improve solutions' feasibility.

3. Since the number of services providing the same function in cloud may be too large, we take steps to improve time performance of the composition approach, including utilizing the notion of skyline and designing parallel running strategy.

The rest of this paper is organized as follows: In section 2, related work is discussed and summarized. In section 3, we define our model for composition and describe the problem. Our approach consisting of service evaluation method, service selection algorithm and running strategy is presented in section 4. Section 5 shows the simulation results. Finally, section 6 gives conclusions and an outlook on possible continuations of our work.

2. Related Work

QoS is a fundamental notion of cloud computing. Many researchers propose QoS obtaining and management technologies for web services, such as client feedback and routing of service operational events [4]. Zibin Zheng et al. [5] conduct several large-scale QoS evaluations on real-world web services. The work in [6] presents a formal model for predicting the availability of web services. In [7] , the authors present a QoS management model for cloud computing based on Fuzzy Cognitive Map (FCM).

In previous studies, different QoS-based web service composition strategies are proposed. Mahammad Alrifai et al. [8][9] define the problem as a multi-dimension multi-choice knapsack problem and address the problem by combining global optimization with local selection methods. By decomposing the optimization problem into small sub-problems, their methods are able to solve the problem in a distributed manner. The work in [10] extends the methods above. The authors present a strategy to further reduce the search space by examining only subsets of the candidate services since the number of candidate services for a composition may be too large. Some researchers use heuristic algorithms to solve the composition problem. Tao Yu et al. [11] model the problem as a multiconstraint optimal path problem. The authors propose a heuristic search method based on the algorithm of single-source shortest paths in directed acyclic graphs. Guobing Zhou et al. [12] propose a planning-based approach that can automatically convert a QoS aware composition task to a planning problem with temporal and numerical features. However, the SLA violation is not considered in above work.

Some studies have been done to solve the SLA violation problem. Undesirable events during runtime of composite service may lead to the violation of temporal QoS constraints. In order to solve this problem, Hussein AI-Helal et al. [13] introduce replaceability as a metric for determining web service composition. The authors also propose a replanning algorithm to handle undesirable events during runtime. Some work establishes runtime adaptation of compositions as a promising tool to achieve SLA satisfactory. In order to minimize the total costs of violations and applied adaptations, Leitner et.al. [14] examine the associated cost tradeoff as an optimization problem in terms of expected QoS values versus the cost of their violations. However, the above strategies are just a kind of remedial measure. The work [15] searches the solution space exhaustively and tries to evaluate the feasibility of all possible combinations. Though the feasibility is the largest, the time complexity is too high. The approach cannot be applied in practice. Adrian Klein et al. [16] propose a network-aware model for service composition in cloud. They consider not only the QoS of web services but also the QoS of network. The authors estimate the network latency between arbitrary network locations of services or users and propose a genetic algorithm to select services that will result in low latency. Although their method is very helpful to minimize execution time violation, there are still some problems. First, the authors assume the network QoS is only relevant to the distance, which is not the truth. Second, their work only focuses on optimizing composite service's execution time. In real applications, QoS constraints defined in a service level agreement usually involve multiple QoS attributes, such as price and availability.

The motivation for our work is to propose an composition approach, guaranteeing solutions' optimality and avoiding SLA violations simultaneously.

3. Problem Description

QoS of single service can be provided by providers, computed based on execution, or collected via users' feedback in terms of the characteristic of each QoS attribute [17]. Our service evaluation method which will be explained in section 4 can be extended to different kinds of QoS attributes. So we just select four QoS attributes which are paid more attention in practice to illustrate the problem here :

1. time (t):The execution duration between the moment when a request is arrived and the moment when the result is obtained.
2. availability (a):The probability that a service is accessible.
3. price (p):The money that the user has to pay to the service provider for the use of service.
4. reputation (r):A measure of services' trustworthiness.

We use QoS vector (<*t*,*a*,*p*,*r*>) to represent these attributes. These QoS attributes can be divided into two subsets: positive attributes and negative attributes. The increase of positive attributes is beneficial for users, such as availability and reputation, whereas the decrease of negative attributes is beneficial for users, such as time and price.

Except the QoS of single services, the QoS of a composite service is also relevant to the structure of composition path. There are four common structures: sequential, parallel, conditional and loop. **Fig. 2** shows the four service composition structures considered in our study. Similar to most works [17][18], aggregation functions for QoS computation of composite services are illustrated in **Table 1**. For example, **Fig. 3** shows a composite service with the function of finding the best used car offers. The users submit their requests to the service. The service then returns a list of best offers along with a credit and an insurance offer for each car on the list. It is composed of four single services with the function of searching for used car information, searching for corresponding credit information, searching for corresponding insurance information and integrating the results respectively. We assume that their QoS vectors are <t_1,a_1,p_1,r_1>, <t_2,a_2,p_2,r_2>, <t_3,a_3,p_3,r_3> and <t_4,a_4,p_4,r_4> respectively. Two structures, sequential and parallel, exist in the composition path. Service 2 and service 3 are parallel. Service 1, the combination of service 2 and service 3, and service 4 are sequential. Then the QoS vector of the composite service is $\left\langle t_1 + \mathrm{Max}(t_2, t_3) + t_4, \prod_{i=1}^{4} a_i, p = \sum_{i=1}^{4} p_i, r = \mathrm{Avg}_{i=1,2,3,4} r_i \right\rangle$.

Today, it is increasingly common for users and cloud providers to agree on explicit service level agreements. According to the above description of QoS, an SLA is currently defined with end-to-end response time, availability, price, and reputation of composite services. We use SLA vector <*St*,*Sa*,*Sp*,*Sr*> to represent the QoS constraints defined in SLA. The positive QoS attributes of a composite service should not be smaller than the corresponding SLA constraints, and the negative QoS attributes of a composite service should not be larger than the corresponding SLA constraints. A composite service is said to be feasible only if all its QoS attributes satisfy the corresponding SLA constraints.

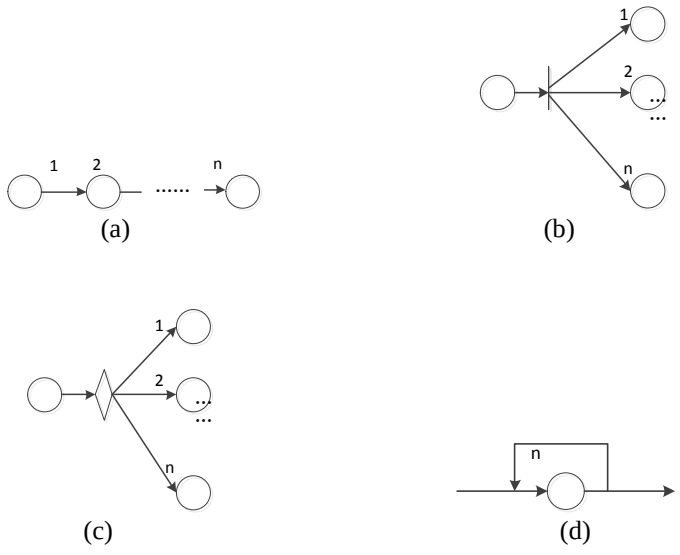


Fig. 2. Service composition structures. (a) sequential; (b) parallel; (c) conditional; (d) loop

Table 1. Aggregation functions for QoS computation

structure	**time (t)**	**availability (a)**	**price (p)**	**reputation (r)**
sequential	$t = \sum_{i=1}^{n} t_i$	$a = \prod_{i=1}^{n} a_i$	$p = \sum_{i=1}^{n} p_i$	$r = \mathrm{Avg}_{i=1,2,..n} r_i$
parallel	$t = \mathrm{Max}_{i=1,2,..n} t_i$	$a = \prod_{i=1}^{n} a_i$	$p = \sum_{i=1}^{n} p_i$	$r = \mathrm{Avg}_{i=1,2,..n} r_i$
conditional	$t = \mathrm{Avg}_{i=1,2,..n} t_i$	$a = \mathrm{Avg}_{i=1,2,..n} a_i$	$p = \mathrm{Avg}_{i=1,2,..n} p_i$	$r = \mathrm{Avg}_{i=1,2,..n} r_i$
loop	$t = \mathrm{n}t_i$	$a = (a_i)^{\mathrm{n}}$	$p = \mathrm{n}p_i$	$r = r_i$

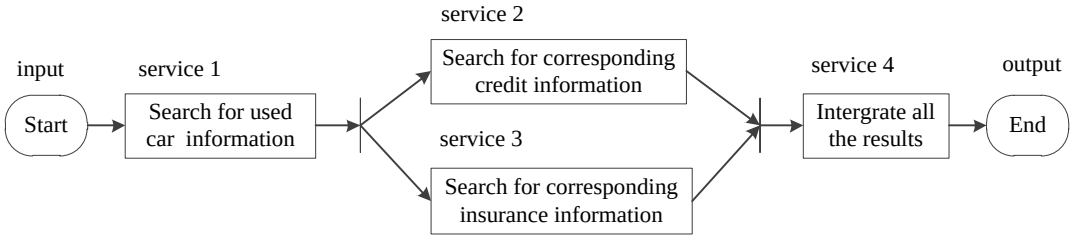


Fig. 3. Example for QoS computation

Based on the above concepts, we model the process of service composition in cloud environment as **Fig. 4**. Through logical decomposition, a user request is translated into a workflow *WFlow* = {*T*, *L*}. *T* = {$task_1$, $task_2$, ...$task_n$} is a set of n tasks, and *L* is a set of relationship between tasks (e.g. sequential, parallel and conditional). *WFlow* defines the component functions and their relationships. In the phase of service discovery, a service set *S* = {s_1, s_2, ...s_n} for each task is discovered according to the functional description of atomic services. **Atomic service** (s_i) is an independent unit to solve a particular task. Each atomic service is related to a QoS vector <t_i,a_i,p_i,r_i>. A **service set** (*S*) is a collection of atomic services with the same function but different QoS attributes. Atomic services in the same service set can substitute for each other in function. Thus, there are many ways of combinations to compose the service. Though these combinations achieve the same function, their QoS are different from each other. Then in the phase of service selection, based on the QoS vector of atomic services and SLA vector, the service selection strategy selects an atomic service from each service set to form the final composition plan.

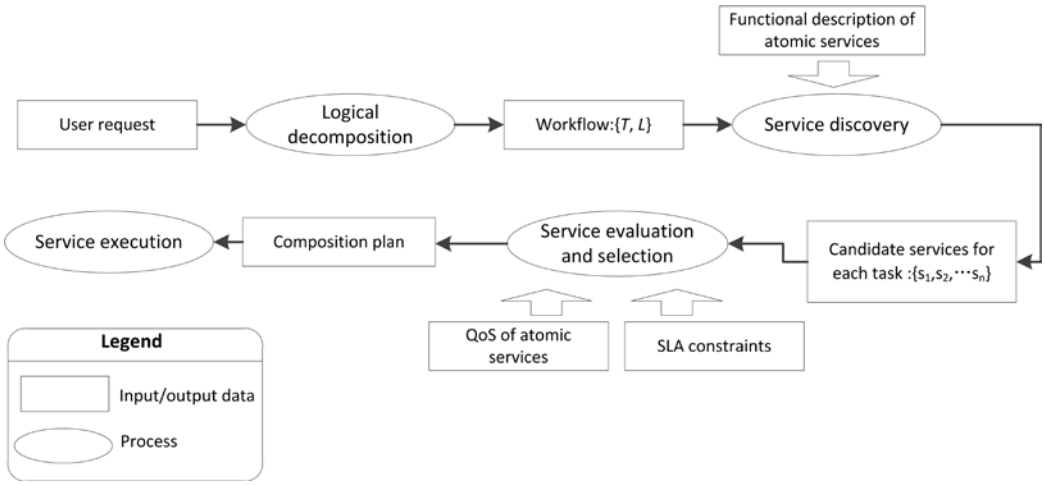


Fig. 4. Process of service composition in cloud environment

Finally, the QoS-based web service composition in cloud environment is modeled as an optimization problem, in which the overall QoS need to be optimized and each QoS attribute need to satisfy the corresponding constraint defined in SLA.

4. Service composition Approach

4.1 Service Evaluation Method

In order to optimize the overall QoS of composite services, we need an evaluation model first. As there are more than one QoS attributes for each composite service, the evaluation model need to transform all the QoS attributes to a comprehensive value.

Given the aim of satisfying SLA constraints, we design the evaluation model based on the following two principles. First, the evaluation model should reflect the solutions' feasibility. For example, the evaluation value of composite service which is feasible should be larger than that of unfeasible composite service. Second, as the QoS attributes of an atomic service may change from original estimations during execution time because of some unexpected situations, such as server overload [15], the original feasible composition plan may become unfeasible after execution phase. In order to solve this problem, we prefer the feasible composite service whose QoS attributes keep longer distance from SLA constraints. So the longer the distance, the more preferable the composite service is.

Based on the above principles, we propose an evaluation method consisting of two parts as follows:

4.1.1 Determining attribute level

Attribute level is determined for each QoS attribute of composite service before the evaluation value of composite service is computed. ς^j in (1) represents the relative distance from the j-th QoS attribute to corresponding SLA constraint, Q^j is the j-th QoS attribute and Sq^j is the corresponding SLA constraint. If the QoS attribute satisfies the corresponding SLA constraint, then $\varsigma^j > 0$. If the QoS attribute equals to the corresponding SLA constraint, then $\varsigma^j = 0$.

Otherwise, $\varsigma^{j}<0$. We divide QoS attributes into three attribute levels according to the relative distances: offgrade attribute, risky attribute and eligible attribute. Offgrade attributes are QoS attributes violating corresponding SLA constraints ($\varsigma<0$). Risky attributes are QoS attributes satisfying QoS constraints but very close to corresponding SLA constraints ($0 \le \varsigma \le 0.1$). Others are eligible attributes ($0.1 < \varsigma$).

$$\varsigma^{j}=\begin{cases}\dfrac{Sq^{j}-Q^{j}}{Sq^{j}}, & Q^{j} \in \text{negative attributes} \\ \dfrac{Q^{j}-Sq^{j}}{Sq^{j}}, & Q^{j} \in \text{positive attributes}\end{cases} \tag{1}$$

4.1.2 Determining attribute level

For composite service whose QoS attributes are all eligible attributes, the evaluation value is computed as (2). α^{j} reflects the user's preference for the j-th QoS attribute ($0 \le \alpha^{j} \le 1, \sum_{j=1}^{o}\alpha^{j}=1$) and the number of QoS attributes is *o*.

$$f^{e}=\sum_{j=1}^{o}\alpha^{j}\times\varsigma^{j} \tag{2}$$

Obviously, composite service with one or more offgrade attributes is not the solution we want to obtain. A penalty operation to it is needed. A valid way to penalize it is to decrease its evaluation value. Risky attribute is a potential inducement to SLA violation which make it unfavorable. We also need to penalize composite service with one or more risky attributes by decreasing its evaluation value. Our penalty strategy is designed based on two considerations: First, the evaluation value of composite service with one or more offgrade attributes should be low enough for fear that this composite service is chosen to be the final output to user. Second, the evaluation value of composite service with one or more risky attributes should be decreased appropriately in order to reduce the probability that this composite service is chosen to be the final solution.

Given the above penalty considerations, the penalty strategy is designed as follows: If a composite service has one or more offgrade attributes, its evaluation value is always -1. To feasible composite service which has one or more risky QoS attributes, the computation of its evaluation value need to multiply a penalty coefficient, which is shown in (3).

$$f^{r}=\left(\sum_{j=1}^{o}\alpha^{j}\times\varsigma^{j}\right)\times\beta,\ 0\le\beta<1 \tag{3}$$

The complete flowchart of service evaluation process is shown in **Fig. 5.**

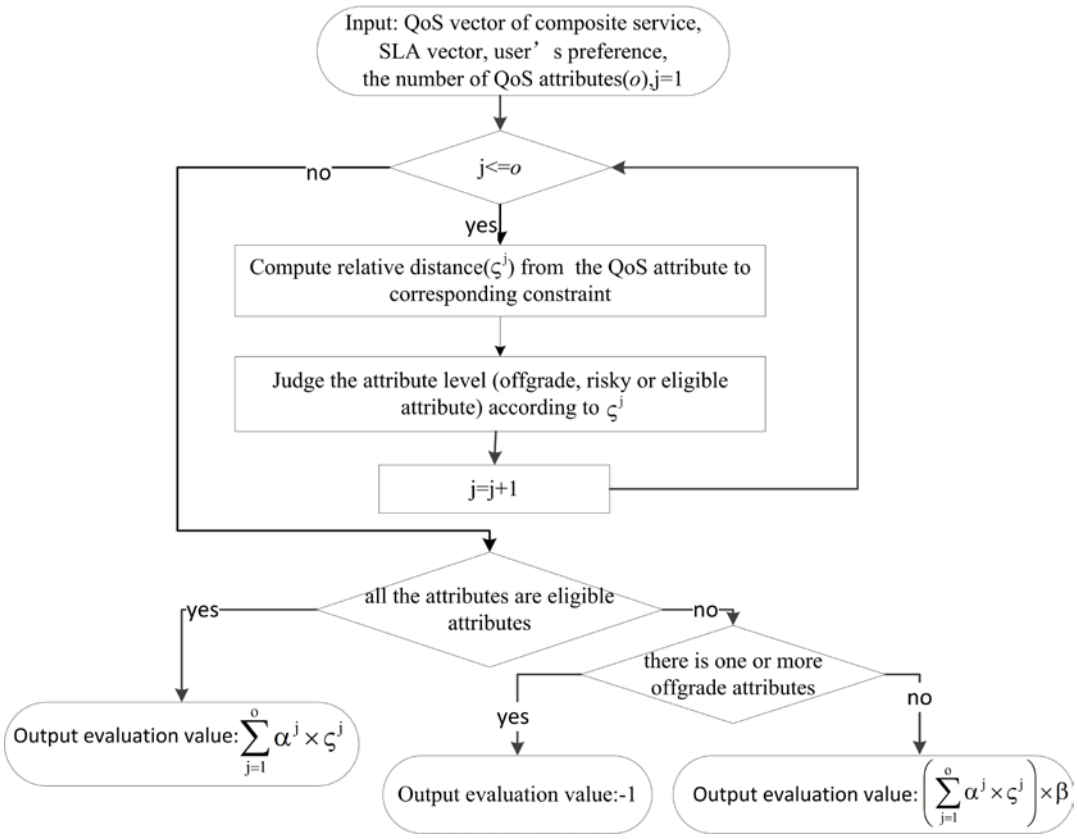


Fig. 5. Flowchart of service evaluation process

4.2 Service Selection Algorithm

The number of atomic services may be extreme large in cloud environment. It will take a lot of time if we try to find the optimal solution by traversing all the combinations. This is unacceptable to users. To solve this problem, we design a heuristic search method based on a genetic algorithm (GA) and simulated annealing algorithm (SA).

GA is population-based approach to search near-optimal solutions. It has been used to solve service composition problem in many work and has been proved to be effective. However, its convergence is not easily controlled. In order to overcome the disadvantages of GA, we combine GA with SA. SA performs better in avoiding locally optimal solution. We utilize the advantages of GA and SA to produce a service selection algorithm. The process of the algorithm is shown in **Table 2**.

In Step2, composite services are encoded as genomes. Each gene in the genome represents the atomic service for each task. An example is shown in **Fig. 6**, the shown genome is corresponding to a composite service that consists of four tasks. Each gene represents the chosen atomic service from each service set. For example, atomic service s_{1i} is chosen to implement $task_1$. In this example, the workflow consists of four tasks, so the genome consists of four genes.

Table 2. Process of service selection algorithm

Service selection algorithm
Step1: Definition and Initialization (capital letters represent constant and lowercase letters represent variable): T_0: initial temperature. T_t: terminal temperature. C: cooling coefficient. K_{max}: the maximal number of iterations under each temperature. M: the size of population. N: constant tag. t: the current temperature, $t=T_0$. k: the number of iterations under current temperature, $k=0$. ρ : the current optimal solution. $f(x)$: the fitness value of individual x.
Step2: Encoding and generating the initial population with M individuals.
Step3: Calculating the fitness values of the individuals in current population. Assigning the individual with the largest fitness value to ρ.
Step4: Executing selection, crossover and mutation operators and then generating a new population.
Step5: If $k < K_{max}$, then $k = k + 1$, calculating the fitness values of the individuals in current population, go to step4. Else, go to step6.
Step6: Selecting the individual (x) with the largest fitness value in the new population, if $f(x) > f(\rho)$, then assign x to ρ.
Step7: If ρ is not updated for continuous N iterations, the algorithm ends and outputs ρ. Else, then $t = C \times t$, go to step8.
Step8: If $t > T_t$, then $k = 0$, go to step4. Else, the algorithm ends and fails to search the solution.

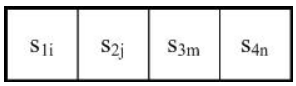


Fig. 6. Example of genome

In the second step, a predefined number (M) of genomes are generated to form the initial population. We adopt the generation method based on the notion of skyline, which we proposed earlier. Please refer to [19] for details. We give a brief introduction of it here. Generally, the initial population of genetic algorithm is generated randomly. In order to improve the solution's quality and convergence speed, we generate one fifth of the initial population based on the notion of skyline and four fifths of the initial population in random. **Fig. 7** shows an example of the notion of skyline. For simplicity, we only use two QoS attributes of atomic services in this example. They are availability and reputation which are both positive attributes. Atomic service d dominates atomic service b because that availability and reputation of d are both better than those of b. Atomic service f is not dominated by any other atomic services because there is no atomic service whose availability and reputation are both better than those of f. In other words, f is non-dominated. Skyline set is composed of atomic services that are non-dominated in a service set. The component atomic services of one fifth individuals in the initial population are chosen from skyline set. Though the computation cost for skyline set can be expensive if the number of atomic services per service set is too large, the computation cost of skyline sets has no influence on the process time of service composition. This is because the process of determining the skyline set does not need to be conducted during service composition time. The skyline set can be determined in advance and stored in caches, and can be updated when there are changes of atomic services.

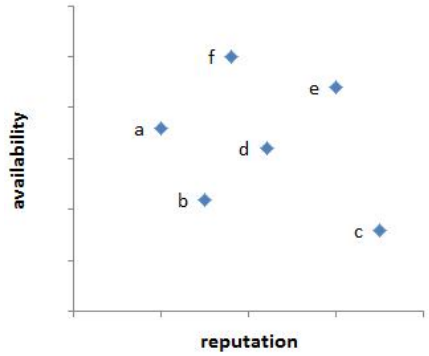


Fig. 7. Example of skyline

In Step3 and Step5, the algorithm evaluates the genome using fitness function. In order to obtain the optimal QoS, the fitness function must promote the increase of positive attributes and the decrease of negative attributes. In addition, the fitness function must penalize solutions that violating SLA constraints. Given the above considerations, we adopt the service evaluation method we proposed in section 4.1 as the fitness function.

In Step4, the generation of new population uses the selection, crossover and mutation operations. The strategy of roulette-wheel selection is used to select individuals to be reproduced via mutation and crossover. The probability that an individual with fitness value $f(s_k)$is chosen from the population is computed as (4).

$$p(s_k) = f(s_k) \Big/ \sum_{j=1}^{M} f(s_j) \tag{4}$$

Where $f(s_j)$ is the fitness value of the j-th individual. Roulette-wheel selection ensures that better individuals have higher chance to be chosen and better genes have higher probability to be inherited. The mutation operator selects a gene of the genome and randomly replaces it with an atomic service in the corresponding service set. In the design of crossover operator, some pairs of the matching genomes are chosen to be combined to generate offspring. For each parent genome, it is divided into two parts by a cut-off point. Then the tail parts are exchanged. The new genome generated by crossover operator is adopted as an individual in the next population with a specified probability. The adoption strategy of new individuals is described as follows:

1. Assuming that two individuals x_i , x_j are chosen to execute crossover operation. After the crossover operation, two new individuals x'_i and x'_j are generated. Their fitness values are $f(x'_i)$ and $f(x'_j)$, respectively. According to the principle of SA, if $\min\{1,\exp(-(f(x_j)-f(x'_j))/t)\}>R$,then x'_j is adopted as an individual in the new population, else, x_j is adopted as an individual in the new population. If $\min\{1,\exp(-(f(x_i)-f(x'_i))/t)\}>R$, then x'_i is adopted as an individual in the new population, else, x_i is adopted as an individual in the new population. R is a random number ranging from 0 to 1.

2. Assuming that individual x_i is chosen to execute mutation operation. After the mutation operation, a new individual x'_i is generated. According to the principle of SA, if $\min\{1,\exp(-(f(x_i)-f(x'_i))/t)\}>R$, then x'_i is adopted as an individual in the new population, else, x_i is adopted as an individual in the new population. R is a random number ranging from 0 to 1.

The above method of generating new population can overcome the premature disadvantage of GA. The traditional GA only adopts new individuals which are better than the current

individuals. So GA is easily to be premature. In our method, new individuals with smaller fitness value also have chance to be adopted. This is helpful to avoid outputting locally optimal solution.

In Step6, the optimal individual under current temperature is selected. In Step 7, if the optimal solution is not changed for continuous N times, then algorithm ends and output the solution. Else, the temperature is reduced.

Step 8 means that step4 to step7 are repeated until the temperature reduces to the terminal temperature.

4.3 Running Strategy

In order to obtain a near-optimal solution through genetic algorithm, it is better to expand the size of initial population. However, the algorithm not only need to execute selection, crossover and mutation operations to the large population, but also needs to compute the fitness value for every individual, which results in the reduction of the approach's efficiency. Given the above consideration, we choose a parallel running strategy.

An important feature of genetic algorithm is that it is easy to be operated in parallel manner. So we transform the algorithm shown in **Table 2** into a parallel approach. The principle is that we divide the initial population into multiple sub populations. These sub populations evolve independently according to the algorithm shown in **Table 2**. However, the evolutionary parameters for different sub populations (such as mutation probability, cooling coefficient) are different. Every time the sub populations reach to final states under certain temperature (Step 6 in **Table 2**), the global optimal solution will be chosen to replace the worst solution of each sub population. The detail design of the parallel approach is shown in **Fig. 8**.

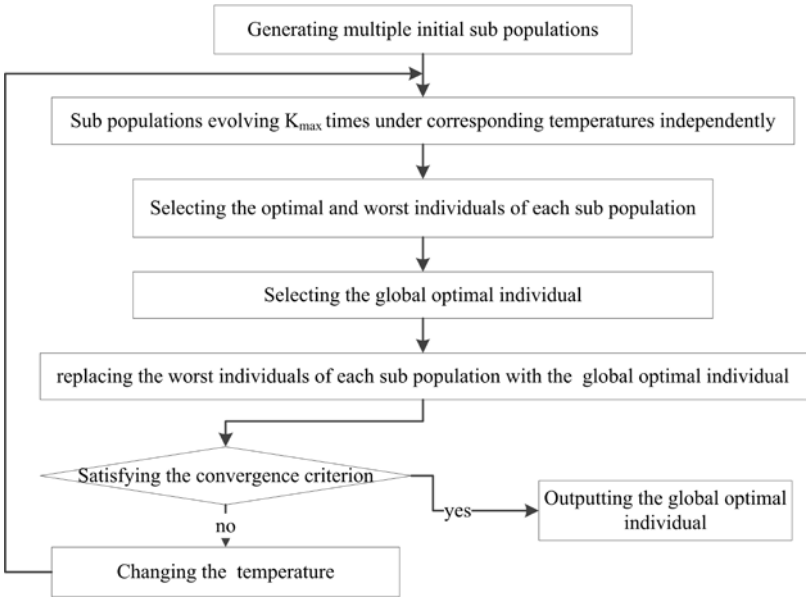


Fig. 8. Overview of the parallel approach

In **Fig. 8**, K_{max} is the maximal number of iterations under each temperature, which has been defined in **Table 2**. The values of K_{max}, T_0, N and M which are defined in **Table 2** for different sub population are equal respectively. The T_t is designed according to T_0 and C to guarantee that the largest number of temperature reduction for every sub population is the same. The convergence criterion in **Fig. 8** is that the global optimal individual does not change for

continuous N iterations. In the parallel approach, the global optimal individual spreads among sub populations every time the sub populations evolve K_{max} times. Considering that the evolution speeds for sub populations are different, we adopt a synchronous processing mechanism, i.e. the sub populations who accomplish K_{max} evolutions ahead switch to waiting status. The global optimal individual will not be selected until all the sub populations accomplish evolution. If there is a sub population converges ahead, its output will still participate the selection of global optimal individual in the following iterations of the parallel approach.

5. Simulation

In this section, we present simulations of our approach. We first describe the setup of our simulation, and then evaluate the search performance and feasibility of our approach: (a) search performance, in terms of the solutions' fitness values and the corresponding runtime required to find the solutions, (b) feasibility, in terms of whether a solution satisfies the SLA constraints or not.

5.1 Simulation Setup

All simulation results were measured with a Sun Java SE 7 VM running on Windows 7 PC with an AMD 4.1 GHz CPU (4 cores) and 8GB memory space.

For simulation, the four QoS attributes mentioned above were considered and they gained equal preference from users. QoS of atomic services were generated randomly. The random values of response time in each service set had a Gaussian distribution and a mean value within the range [20, 1500] millisecond. The ranges of availability, price and reputation were [0.95, 1], [2, 15] dollar and [0.4, 1] in order. We then created SLA vectors of four values in terms of the number of service sets. Each SLA vector, which is noted by $<St,Sa,Sp,Sr>$, is corresponding to one SLA-based composition request. In reality, QoS constraints defined in SLA are generated through the negotiation between users and service providers. In our simulation, in order to evaluate our approach reasonably and efficiently, we generated the QoS vectors according to the number of service sets and the value ranges of QoS attributes. They were computed as follows. m is the number of service sets required by the composition request.

$$St = 760m + 260(m + 1) \tag{5}$$

$$Sa = 0.98m \tag{6}$$

$$Sp = 8m \tag{7}$$

$$Sr = 0.72 \tag{8}$$

For the purpose of our simulation, the following approaches were used.

1. GA_S: It is an approach based on our earlier proposed generic algorithm [19], which is adapted to the new problem described in this work. Its population size is 150.

2. GA_SA: It is the new approach described in **Table 2** of this paper. However, it does not adopt the new proposed running strategy described in section 4.3. The maximal number of iterations under each temperature is 200. The mutation probability is 0.05.

3. GA_SA_P(K_{max} =200): It is the new approach described in section 4. It is notable that it adopts the parallel running strategy proposed in section 4.3. The maximal number of iterations

under each temperature is 200. The initial population was divided into three sub populations. The evolutionary parameters for sub populations are given in **Table 3**.

4. GA _SA_P(K_{max} =400): It is the new approach described in section 4. It is notable that it adopts the parallel running strategy proposed in section 4.3. The maximal number of iterations under each temperature is 400. The initial population was divided into three sub populations. The evolutionary parameters for sub populations are given in **Table 3**.

5. MCSP: It is an algorithm proposed in [11] to solve the problem of QoS and constraints aware service composition. It is based on the algorithm of single-source shortest paths in directed acyclic graphs. MCSP is able to find a composition path that produces the highest utility subject to the multiple QoS constraints because it topologically visits all nodes in the service candidate graph. However, it is very time consuming.

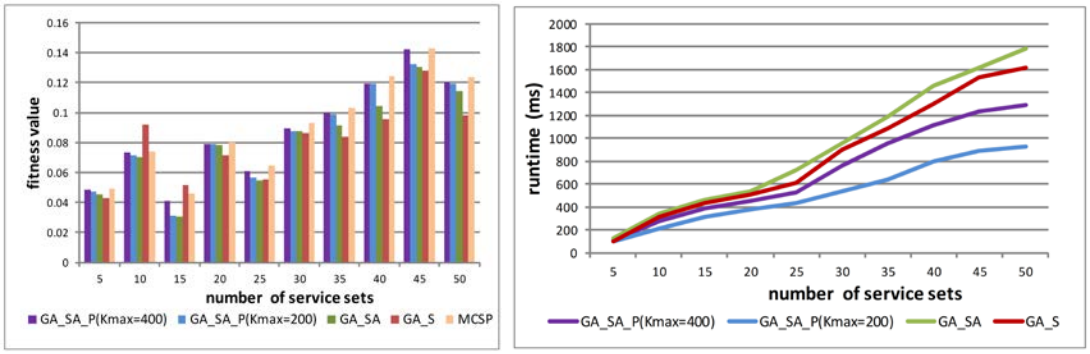
Table 3. Setting of evolutionary parameters for sub populations

Parameter	Sub population 1	Sub population 2	Sub population 3
T_0	0.5	0.5	0.5
C	0.98	0.95	0.93
T_t	2.41×10^{-2}	2.28×10^{-4}	9.36×10^{-6}
M	50	50	50
Mutation probability	0.05	0.1	0.02

The convergent criterion for all the algorithms was that the best fitness value does not improve over the last 10 iterations (N=10).

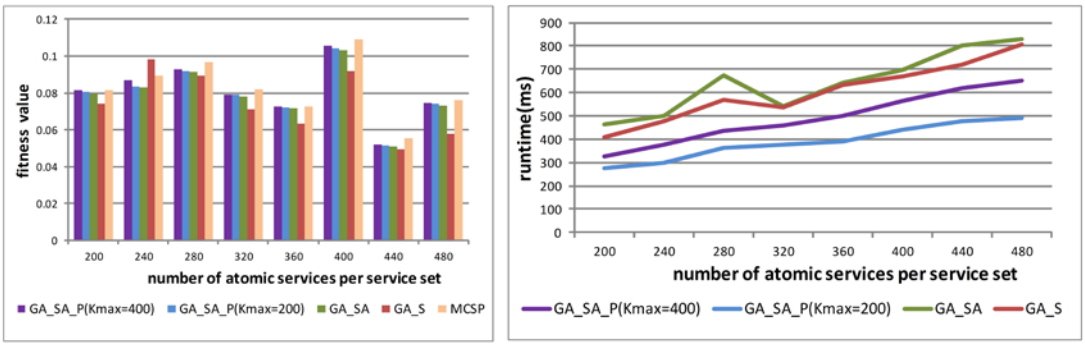
5.2 Search Performance

To evaluate the search performance of our approach, we compare the solutions' fitness values and runtime of GA_SA_P(K_{max} =400), GA_SA_P(K_{max} =200), GA_SA, GA_S and MCSP versus an increasing problem size. **Fig. 9** plot the fitness value and runtime against an increasing number of service sets with a fixed number (320) of atomic services per service set. **Fig. 10** plot the fitness value and runtime against an increasing number of atomic services per service set with a fixed number (20) of service sets.



(a) (b)

Fig. 9. Fitness value and Runtime VS number of service sets



(a) (b)

Fig. 10. Fitness value and Runtime VS number of atomic services per set

From **Fig. 9(a)** and **Fig. 10(a)**, we can see that in most scenarios the fitness values generated by GA_SA were much higher than those generated by GA_S. The contrast was more obvious when the problem scale become larger. This is because GA_SA can avoid locally optimal solution effectively, whereas GA_S is easily to be premature. In some scenarios where GA_S avoided the premature feature, the fitness values generated by GA_SA and GA_S were at the same level. We can also see that the fitness values generated by GA_S were higher than those generated by GA_SA in only a few scenarios. We validated the feasibility of the corresponding solutions and found that only one solution was feasible, which indicated that GA_S attempted to obtain the large fitness value at the cost of feasibility. **Fig. 9(a)** and **Fig. 10(a)** also show that the fitness values generated by GA_SA_P(K_{max}=200) were a little higher than those generated by GA_SA. This is because that the global optimal solution spreads among sub populations periodically in the parallel strategy, which helps sub populations to get rid of local convergence. Furthermore, the fitness values generated by GA_SA_P(K_{max}=400) are very close to those generated by MCSP. MCSP generates the optimal solutions because it topologically visits all nodes in the service candidate graph. Thus, the simulation results indicate that GA_SA_P can generate near-optimal solutions when K_{max} is large.

Fig. 9(b) and **Fig. 10(b)** illustrate the runtime of four approaches versus an increasing dataset size. The runtime of MCSP is extremely large and over ten times as much as other approaches. Thus, the runtime of MCSP is not shown in the figure. From the figure we can see that the runtime of GA_SA was a little larger than that of GA_S and the gap was not obvious. However, the feasibility and fitness values of solutions generated by GA_SA were better than those of GA_S. Therefore, it seems that GA_S is just a little faster, because it fails to improve the feasibility and optimality of its solution. The runtime of GA_SA_P(K_{max}=200) was much lower than that of GA_SA. Obviously the parallel running strategy results in high efficiency.

It's worth noting that the fitness values generated by GA_SA_P(K_{max}=400) was a little higher that those generated by GA_SA_P(K_{max}=200) while the running time of GA_SA_P(K_{max}=400) was higher than those of GA_SA_P(K_{max}=200) in the figures. This illustrate that the maximal number of iterations under each temperature can influence the performance of the parallel strategy obviously. The larger the K_{max} is, the higher the probability of obtaining the optimal solution is. However, with the increasing of K_{max}, the runtime of the approach increases. So in real case a compromise is needed.

5.3 Feasibility Evaluation

In this simulation, we measured the feasibility of our approach. For this purpose, we created 80 scenarios. In the first 40 scenarios, the number of atomic services per service set was

assigned a fix number 320 and the number of service sets ranged from 2 to 41. In the last 40 scenarios, the number of atomic services per service set increased from 40 to 820 at a rate of 20 per instance and the number of service sets was assigned a fix number 15. **Table 4** shows the statistical results of the three approaches' feasible rate obtained from 80 instances.

Table 4. Statistical results of feasible evaluation

	GA_SA_P(K_{max}=400)	**GA_SA**	**GA_S**	**MCSP**
The number of feasible solutions	80	79	74	80
Feasible rate	100%	98.75%	92.50%	100%

In the simulation, there was only one scenario where GA_SA failed to search the solution. In the remaining scenarios the solutions generated by GA_SA all satisfied the QoS constraints defined in SLA. From **Table 4**, we observed that the feasible rate of GA_SA was higher than that of GA_S. Through the parallel running strategy, the feasibility can reach to 100%. GA_S output solutions in all the scenarios. However, 7.5% of these solutions were unfeasible. The contrast shows that our proposed service evaluation method reflects the feasibility of services effectively and our approach can be applied to real situations in terms of its high feasibility.

6. Conclusion

In this paper we described a feasibility-enhanced approach to QoS-based web service composition in cloud environment, consisting of a service evaluation method, a service selection algorithm and a parallel running strategy. To deal with the SLA violation problem, our service evaluation method considers SLA constraint as an important factor. Based on the service evaluation method, genetic and simulated annealing algorithm, a service selection algorithm and corresponding running strategy are designed, which achieve a better feasibility without sacrificing solutions' optimality. In our future work, we plan to conduct real-world experiments and optimize our approach. Additionally, we would like to develop a runtime recovery strategy to handle the undesiring events, which is also important to enhance the services' feasibility.

References

[1] M. Bichler and K.J. Lin, "Service-oriented computing," *IEEE Computer*, vol. 39, no. 3, pp. 99–101, 2006. Article (CrossRef Link)

[2] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based internet of things: discovery, query, selection, and on-demand provisioning of Web services," *Services Computing IEEE Transactions on*, vol. 3, no. 3, pp. 223-235, 2010. Article (CrossRef Link)

[3] S. Stein, T.R. Payne, and N.R. Jennings, "Robust execution of service workflows using redundancy and advance reservations," *Services Computing IEEE Transactions on*, vol. 4, no. 2, pp. 125-139, 2011. Article (CrossRef Link)

[4] L. Zeng, H. Lei, and H. Chang, "Monitoring the QoS for Web services," in *Proc. of International Conference on Service-Oriented Computing*, Springer-Verlag, pp. 132-144, 2007. Article (CrossRef Link)

[5] Z. Zheng, Y. Zhang, and M.R. Lyu, "Investigating QoS of real-world Web services," *Services Computing IEEE Transactions on*, vol. 7, no. 1, pp.32-39, 2014. Article (CrossRef Link)

[6] M. Silic, G. Delac, I. Krka, and S. Srbljic, "Scalable and accurate prediction of availability of atomic Web services," *Services Computing IEEE Transactions on*, vol. 7, no. 2, pp. 252-264, 2014. Article (CrossRef Link)
[7] P. Zhang, Z. Yan, "A QoS-aware system for mobile cloud computing," in *Proc. of IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pp. 518-522, 2011. Article (CrossRef Link)
[8] M. Alrifai, T. Risse, P. Dolog, and W. Nejdl, "A scalable approach for qos-based Web service selection," in *Proc. of Service-Oriented Computing–ICSOC 2008 Workshops*, Springer Berlin Heidelberg, pp. 190-199, 2009. Article (CrossRef Link)
[9] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *Proc. of the 18th International Conference on World Wide Web*, pp. 881-890, 2009. Article (CrossRef Link)
[10] M. Alrifai, D. Skoutas, and T. Risse, "Selecting skyline services for qos-based Web service composition," in *Proc. of the 19th International Conference on World Wide Web*, ACM Press, pp. 11–20, 2010. Article (CrossRef Link)
[11] T. Yu, Y. Zhang, K. J. Lin, "Efficient algorithms for Web services selection with end-to-end qos constraints," *ACM Transactions on Web*, vol. 1, no. 1, pp.1 -25, 2007. Article (CrossRef Link)
[12] G. Zou, Q. Lu, Y. Chen, R. Huang, Y. Xu, and Y. Xiang, "QoS-aware dynamic composition of Web services using numerical temporal planning," *Services Computing IEEE Transactions on*, vol. 7, no, 1, pp. 18-31, 2014. Article (CrossRef Link)
[13] H. Al-Helal and R. Gamble, "Introducing replaceability into web service composition," *Services Computing, IEEE Transactions on*, vol. 7, no. 2, pp. 198–209, 2014. Article (CrossRef Link)
[14] P. Leitner, W. Hummer, and S. Dustdar, "Cost-based optimization of service compositions," *Services Computing, IEEE Transactions on*, vol. 6, no. 2, pp. 239–251, 2013. Article (CrossRef Link)
[15] B.Y. Wu, C.H. Chi, and S. Xu, "Service selection model based on qos reference vector," *Services, 2007 IEEE Congress on*, pp. 270–277, 2007. Article (CrossRef Link)
[16] A. Klein, F. Ishikawa, and S. Honiden, "Towards network-aware service composition in the cloud," in *Proc. of 21th International Conference on World Wide Web*, France, pp. 959-968, 2012. Article (CrossRef Link)
[17] Y. Liu, A.H. Ngu, and L.Z. Zeng, "Qos computation and policing in dynamic web service selection," in *Proc. of 13th International World Wide Web Conference on Alternate Track Papers & Posters*, pp.66–73, 2004. Article (CrossRef Link)
[18] J. Xiao and R. Boutaba, "Qos-aware service composition and adaptation in autonomic communication," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 12, pp.2344–2360, 2005. Article (CrossRef Link)
[19] D. Wang, Y. Yang, and Z. Mi, "A genetic-based approach to web service composition in geo-distributed cloud environment," *Computers & Electrical Engineering*, vol. 43, pp. 129–141, 2015. Article (CrossRef Link)

Dandan Wang received the B.S degree in 2012 and is working toward the Ph.D degree at the School of Computer and Communication Engineering at University of Science and Technology Beijing, China. Her main research interests include service-oriented architectures, service composition, and scheduling of distributed resources in the cloud.

Hao Ding received the B.S degree in 2010 from University of Science and Technology Beijing, China. Now he is working toward the Ph.D degree both in the School of Computer and Communication Engineering at University of Science and Technology Beijing, China. His main research interests include network measurement, service optimization, and cloud computing.

Yang Yang received B.S. in aotomation from the Department of Automation, Beijing Institute of Iron and Stell, in 1982. Received his Ph.D. in information Engineering, from University of Science and Technology in Lillie, France, in 1988. He has been a professor of University of Science and Technology Beijing since 1988. From 1994, he has been the senior member of many national and provincial technique committees. Prof. Yang Yang's research interests include service science and cloud computing, intelligent control, image processing and pattern recognition, multimedia communication, grid technology. He has co-authored more than 200 refereed journal and conference papers, and several books.

Zhenqiang Mi received B.S. in aotomation and Ph.D. in communication engineering, both from School of Information Engineering, University of Science and Technology Beijing, in year 2006 and 2011, respectively. From 2011, he is assistant professor with the school of computer and communication engineering, University of Science and Technology Beijing. Prof. Mi is IEEE member, and serves as a frequent reviewer in several international journals and TPC member in several international conferences. His research interest includes service computing, multi-robot systems, connectivity in mobile ad hoc networks and cloud computing in mobile environments.

Li Liu received the Ph.D degree in computer and science from University of Science and Technology Beijing, China, in 2005. She is now an associate professor in University of Science and Technology Beijing. She is a Master graduate tutor. And her research interests are in the areas of distributed systems, network security. She is a member of the IEEE and the ACM.

Zenggang Xiong received his Ph.D. degree in Computer Technology from University of Science and Technology Beijing, China, in 2009. He has been a professor of Hubei Engineering University. His research interest includes computer network, cloud computing, big data and Internet of things.