Service Composition Based on Niching Particle Swarm Optimization in Service Overlay Networks

Jianxin Liao, Yang Liu, Jingyu Wang and Xiaomin Zhu
State Key Lab of Networking and Switching Technology,
Beijing University of Posts and Telecommunications,
Hai Dian, Beijing 100876 - China
[e-mail: jxliao.scholar@gmail.com, {liuyang_5, wangjingyu, zhuxiaomin} @ebupt.com]
*Corresponding author: Jianxin Liao

Received October 28, 2011; revised January 9, 2012; accepted February 4, 2012; published April 25, 2012

Abstract

Service oriented architecture (SOA) lends itself to model the application components to coarse-grained services in such a way that the composition of different services could be feasible. Service composition fulfills numerous service requirements by constructing composite applications with various services. As it is the case in many real-world applications, different users have diverse QoS demands issuing for composite applications. In this paper, we present a service composition framework for a typical service overlay network (SON) considering both multiple QoS constraints and load balancing factors. Moreover, a service selection algorithm based on niching technique and particle swarm optimization (PSO) is proposed for the service composition problem. It supports optimization problems with multiple constraints and objective functions, whether linear or nonlinear. Simulation results show that the proposed algorithm results in an acceptable level of efficiency regarding the service composition objective under different circumstances.

Keywords: Service composition, SON, particle swarm optimization, niching technique, multi-constraint optimal service composition path

This work was jointly supported by: (1) the National Basic Research Program of China (No. 2012CB315802); (2) National Natural Science Foundation of China (No. 61072057, 60902051, 61101119, 61121001); (3) PCSIRT (No. IRT1049); (4) National Key Science & Technology Specific Project of China (No. 2011ZX03002-001-01, Research about Architecture of Mobile Internet).

http://dx.doi.org/10.3837/tiis.2012.04.009

1. Introduction

With the development of advanced computing and communication technologies, users are pursuing various QoS-aware applications aiming for their numerous diverse demands. However, the conventional client-server mechanism, which is inefficient, unreliable and inflexible, is inadequate to fulfill nowadays users' personalized requirements. These challenges motivate the emergence of service oriented architecture (SOA), in which application components are modulated to coarse-grained services to achieve reusability and adaptability. The service provision in SOA is classified into two categories: single-service application and multi-service application. For a single-service application, the system searches for all services qualified for the functional demands and select the optimal one for execution. For a multi-service application, the system invokes several services in a specific order and constructs a composite application for execution, which is defined as service composition.

Service composition can be divided into three steps in process: 1) submission of composite request; 2) service selection; 3) service execution and result acquirement. In the first step, the user submits the functional descriptions like the target of the composite application and the non-functional specification like the QoS constraints of the composite application. The composite application can be modeled as a workflow or a directed attribute graph (DAG). In the second step, the system selects the optimal services based on the QoS constraints, e.g., delay, cost, reliability. Then the system invokes these services and returns the result to the user in the last step, which is out of our concern.

To the best of our knowledge, most existing solutions only discuss specific service composition issues, and ignore the users' requirements and load balance. Therefore, we focus on a more general service composition model considering not only multiple QoS constraints but also load balance factors. To solve this problem, we propose a service selection algorithm Niching PSO which can support multi-constraint multi-objective problems, whether linear or nonlinear. The simulation results show that the service selection algorithm is efficient and effective for service composition.

The rest of this paper is organized as follows. Section 2 presents the system model and problem formulation. The proposed algorithm is described in Section 3. Section 4 presents the simulation results, and analyzes the performance of the proposed algorithm. Section 5 presents the related work. Finally, Section 6 concludes this paper.

2. System Model and Problem Formulation

In this section, we first introduce our system environment. Then we present our system architecture. At the end of this section, we formally define our service selection problem regard to QoS constraints and load balance factors.

2.1 System Environment

In a service overlay network (SON) [24], service nodes (SNs) provide a large number of services for various functions. Duplicated services may be provided by diverse service nodes. Service nodes are attached to different physical nodes, e.g., servers, laptops, mobile phones etc. Service nodes intercommunicate with each other via overlay links. An overlay link consists of one or more physical connections on the lower layer. The services are registered in the service directory. Because of variant performance criteria concerning the service nodes, QoS

parameters of the same service implemented on different service nodes are not the same. When service composition is carried on, the service directory lookups and selects services based on specific QoS rules.

The service composition process is depicted in **Fig. 1**. First, the user client submits the composition request to the gateway of SON (step 1). The request contains the information about the composited application, e.g., tasks of the application and the correlation between them, input and output parameters of tasks, QoS requirements of the composited application. The gateway forwards the request to the service directory (step 2). The service directory searches among the services for each task based on its functional descriptions. Then, the service directory computes the optimal service selection strategy based on non-functional properties of tasks and invokes services hop by hop until the last service is executed (step3-5). At last, the service composition result is returned to the user via the gateway (step6-7).

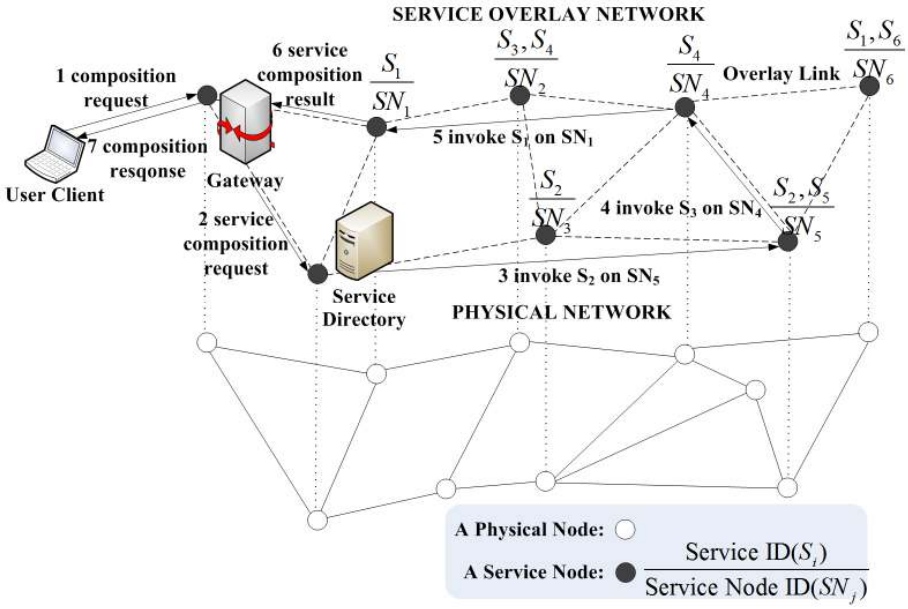


Fig. 1. System Environment

2.2 System Architecture

In this section, we explain SON in detail. To specify service composition, we modified Fig. 1 and represented a service composition example in **Fig. 2**. As shown in **Fig. 2-(a)**, the functional profile of a composition request can be denoted by a task set $TS = \{T_1, T_2, ..., T_a, T_M\}$, in which T_a is the a[th] task of the application and M is the total task number. As shown in **Fig. 2-(a)**, tasks T_1 and T_4 have sequential order in the task set. Tasks T_2 and T_3 have parallel order, whereas tasks T_5 and T_6 have selective order. The details of service composition topologies are further illustrated in Section 2.3.2. When the composition request is forwarded to the service directory, the search and match process is carried out based on the task set. As shown in **Fig. 2-(b)**, if all tasks have been matched with proper services, the service directory transforms the task set to service sets $SSs = \{SS_1, SS_2, ..., SS_i, SS_M\}$, in which SS_i is the corresponding service set of service S_i for the task T_a. There are several service instances in each service set. These service instances have the same function and different non-functional parameters. They may be implemented on different service nodes. Finally, the service directory selects the optimal

service composition path (SCP) from all candidate paths. As shown in **Fig. 2-(c)**, $SCPI\{S_1I_1, S_2I_1, S_3I_1, S_4I_1, S_6I_1\}$ and $SCPII\{S_1I_1, S_2I_1, S_3I_1, S_4I_2, S_5I_1\}$ are two candidate paths for service composition. In this figure, S_iI_k means the k[th] instance of service i. For two candidate SCPs, not only service instance sets are different, but also overlay links between service instances are not the same. Suppose there are M services in an application and N_i Instances for each service i, there turns to be $\prod_{i=1}^{M} N_i$ candidate SCPs for service composition.

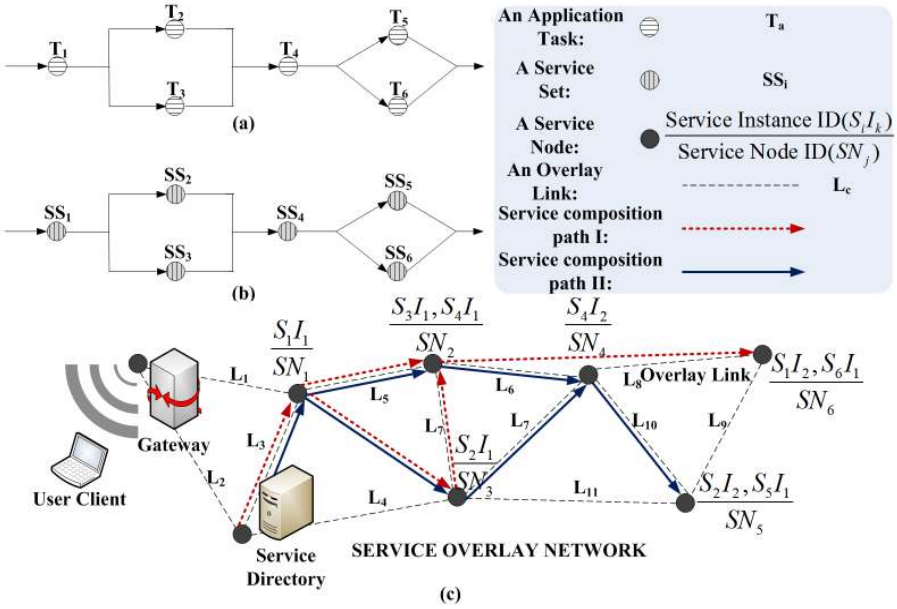


Fig. 2. System Architecture

First, the service directory chooses service instances with the same function for each service set. This process produces plenty of different candidate SCPs. They have the same topology with the task set. Since the task set often possesses several baseline topologies (like parallel and selective), candidate SCPs can't be handled by optimization algorithms directly. We propose a mechanism to transform basic topologies to the simple sequence topology. Therefore, topologies of candidate SCPs can be transformed to sequence topologies. Then the service selection algorithm selects the optimal SCP from all possible candidate SCPs considering load balance (e.g. CPU occupation rate and bandwidth occupation rate) and multiple constraints (e.g. cost, delay and reliability).

2.3 Problem Definition

In this section, we first discuss load balance and QoS parameters of service instances, and then specify service composition topologies and the influence to attributes of SCPs. At last, we present the formal definition of service selection problem with regard to multi-constraint optimal service composition path (MCOSCP).

2.3.1 Load Balance and QoS Parameters

In this section, we refer to two load balance parameters and three QoS parameters of service instances.

CPU occupation rate (*Cor*) is the parameter depicting the percentage of the computing

ability required by a service instance in the available computing ability of the corresponding SN. *Cor* can be calculated using $Cor = CPU_{S_iI_k}^{required} / CPU_{SN_j}^{available}$. $CPU_{S_iI_k}^{required}$ represents the computing ability required by a service instance S_iI_k, which can be estimated when S_iI_k is developed and uploaded to the service directory. $CPU_{SN_j}^{available}$ represents the available computing ability of the SON node SN_j on which the S_iI_k is implemented. It can be detected and advertised periodically by SN_j.

Bandwidth occupation rate (*Bor*) is similar to the *Cor*, which describes the bandwidth's expected usage percentage of overlay links between candidate service instances. These service instances may be implemented on diverse SNs. *Bor* can be calculated by $Bor = BW_{L_c}^{required} / BW_{L_c}^{available}$. $BW_{L_c}^{required}$ represents the bandwidth of L_c required by two candidate service instances. It can be estimated by the output data rate of predecessor service instance or the input data rate of successor service instance. $BW_{L_c}^{available}$ represents the unoccupied bandwidth of L_c, which can be measured by a lightweight approach [18].

Cost (*C*) refers to the price which the service requester pays for the specific service instance. When the service instance is registered, the cost is also uploaded to the service directory in the meantime. If the cost of a service instance alters, the information in the service directory must be kept updated.

Delay (*D*) measures the overall time of the service execution, which is calculated by $D = T_1 + T_2 + T_3$. T_1 is the service execution time which measures the interval between the moment when the service instance starts executing and the moment when the execution is finished. T_2 is the request queuing time which measures the interval between the moment when the service request is arriving at the queue of the corresponding service instance and the moment when the service instance starts to be executed. T_3 is the transporting time which is the sum of transmission time and propagation time.

Reliability (*R*) represents the proportion of the system's available period, which can be calculated by $R = T_{available} / T_{total}$. In this formula, $T_{available}$ is the time when the system works normally, T_{total} is the total observed time. Because practical systems are generally reparable systems, they are in the cycle composed of failure and normal statuses. Consequently, $T_{available}$ is a set of time slices when the service instances in the system are available, i.e. $T_{available} = \sum_{i=1}^{N} TS_i$. Then, the reliability turn to be calculated by $R = \sum_{i=1}^{N} TS_i \Big/ T_{total}$.

2.3.2 Service Composition Topologies

There are four basic topologies of service composition, which is sequence, parallel, selective and loop topologies. These topologies can construct the vast majority of service composition applications. In this section, we describe their conversion to specific service sets and analyze how load balance and QoS parameters are calculated.

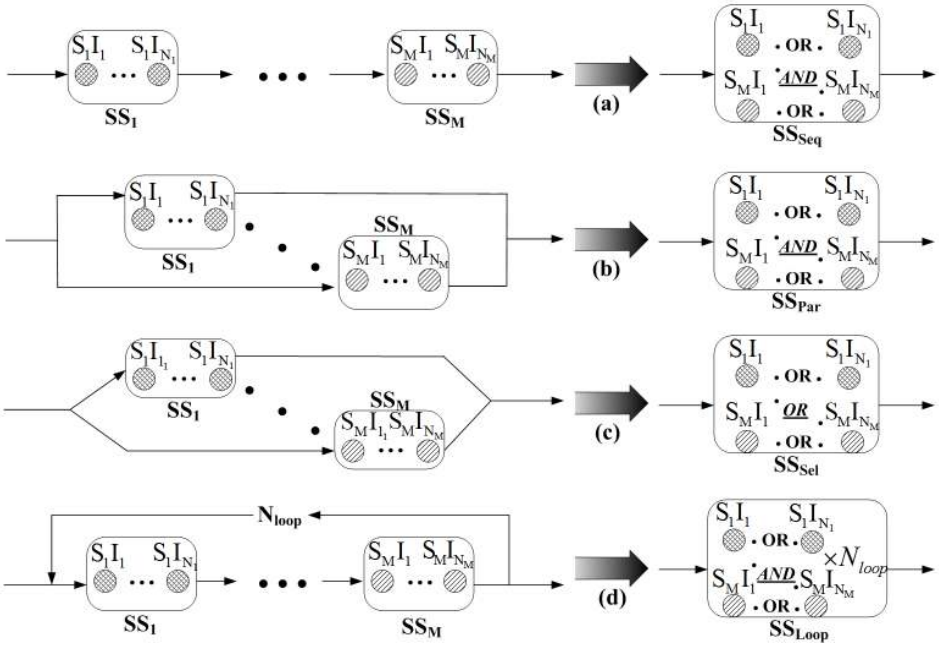


Fig. 3. Service Composition Topologies

Discussion on the Sequence Topology

As shown in **Fig. 3-(a)**, the sequence service composition topology can be converted to SS_{Seq}. In SS_{Seq} shown in the right side of **Fig. 3-(a)**, the service selection is accomplished by selecting any service instance S_iI_k ($i=1,...,M, k=1,...,N_i$) in each service set SS_i. Apparently, there are $\prod_{i=1}^{M} N_i$ candidate SCPs, and each path is a possible solution to the MCOSCP problem. The load balance and QoS parameters of the d[th] SCP is calculated by the formula below:

$$
\begin{aligned}
&C(SCP_d^{Seq}) = \sum_{i=1}^{M} C(S_iI_{k_i}) \\
&D(SCP_d^{Seq}) = \sum_{i=1}^{M} D(S_iI_{k_i}) \\
&R(SCP_d^{Seq}) = \prod_{i=1}^{M} R(S_iI_{k_i}) \\
&Cor(SCP_d^{Seq}) = \max(Cor(S_1I_{k_1}),..., \\
&\qquad Cor(S_iI_{k_i}),..., Cor(S_MI_{k_M})) \\
&Bor(SCP_d^{Seq}) = \max(Bor(L^{in}_{S_1I_{k_1}}),..., \\
&\qquad Bor(L^{in}_{S_iI_{k_i}}),..., Bor(L^{in}_{S_MI_{k_M}}), Bor(L^{out}_{S_MI_{k_M}})) \\
&k_i = 1,...,N_i,\ i=1,...,M,\ d=1,...,\prod_{i=1}^{M} N_i.
\end{aligned}
\tag{1}
$$

where $L^{in}_{S_iI_{k_i}}$ and $L^{out}_{S_iI_{k_i}}$ are respectively the input and output overlay links of service node SN_j on which the service instance $S_iI_{k_i}$ is implemented.

Discussion on the Parallel Topology

As shown in **Fig. 3-(b)**, the parallel service composition topology can be converted to SS_{Par}. In SS_{Par} shown in the right side of **Fig. 3-(b)**, the service selection is accomplished by selecting

any service instance S_iI_k ($i=1,...,M$, $k=1,...,N_i$) in each service set SS_i. This process of SS_{Par} is similar to that of SS_{Seq}. And there are also $\prod_{i=1}^{M} N_i$ candidate SCPs for SS_{Par}. The load balance and QoS parameters of the d[th] SCP is calculated by this formula:

$$\begin{aligned}
&C(SCP_d^{Par}) = \sum_{i=1}^{M} C(S_iI_{k_i}) \\
&D(SCP_d^{Par}) = \max(D(S_1I_{k_1}),..., \\
&\qquad D(S_iI_{k_i}),..., D(S_MI_{k_M})) \\
&R(SCP_d^{Par}) = \prod_{i=1}^{M} R(S_iI_{k_i}) \\
&Cor(SCP_d^{Par}) = \max(Cor(S_1I_{k_1}),..., \\
&\qquad Cor(S_iI_{k_i}),..., Cor(S_MI_{k_M})) \\
&Bor(SCP_d^{Par}) = \max(Bor(L^{in}_{S_1I_{k_1}}), Bor(L^{out}_{S_1I_{k_1}}),..., \\
&\qquad Bor(L^{in}_{S_iI_{k_i}}), Bor(L^{out}_{S_iI_{k_i}}),..., Bor(L^{in}_{S_MI_{k_M}}), Bor(L^{out}_{S_MI_{k_M}})) \\
&k_i = 1,..., N_i,\ i = 1,..., M,\ d = 1,..., \prod_{i=1}^{M} N_i.
\end{aligned} \tag{2}$$

Discussion on the Selective Topology

As shown in **Fig. 3-(c)**, the selective service composition topology can be converted to SS_{Sel}. In SS_{Sel} shown in the right side of **Fig. 3-(c)**, the service selection is accomplished by selecting any service instance S_iI_k ($i=1,...,M$, $k=1,...,N_i$) in any service set SS_i. Each service set SS_i is selected depending on the probability P_{SS_i} ($\sum_{i=1}^{M} P_{SS_i} = 1$). Apparently, there are $\sum_{i=1}^{M} N_i$ candidate SCPs and each path is a possible solution to the MCOSCP problem. The load balance and QoS parameters of the d[th] SCP is calculated by this formula:

$$\begin{aligned}
&C(SCP_d^{Sel}) = \sum_{i=1}^{M} (C(S_iI_{k_i}) \times P_{SS_i}) \\
&D(SCP_d^{Sel}) = \sum_{i=1}^{M} (D(S_iI_{k_i}) \times P_{SS_i}) \\
&R(SCP_d^{Sel}) = \sum_{i=1}^{M} (R(S_iI_{k_i}) \times P_{SS_i}) \\
&Cor(SCP_d^{Sel}) = \sum_{i=1}^{M} (Cor(S_iI_{k_i}) \times P_{SS_i}) \\
&Bor(SCP_d^{Sel}) = \sum_{i=1}^{M} (\max(Bor(L^{in}_{S_iI_{k_i}}), Bor(L^{out}_{S_iI_{k_i}})) \times P_{SS_i}) \\
&k_i = 1,..., N_i,\ i = 1,..., M,\ d = 1,..., \sum_{i=1}^{M} N_i.
\end{aligned} \tag{3}$$

Discussion on the Loop Topology

As shown in **Fig. 3-(d)**, the loop service composition topology can be converted to SS_{Loop}. In SS_{Loop} shown in the right side of **Fig. 3-(d)**, the service selection is accomplished by selecting any service instance S_iI_k ($i=1,...,M$, $k=1,...,N_i$) in each service set SS_i. The selected service instances will be executed N_{loop} times in the whole service composition process. The number of candidate SCPs of SS_{Loop} is the same with that of SS_{Seq}, i.e. $\prod_{i=1}^{M} N_i$. The load balance and QoS parameters of the d[th] SCP is calculated by this formula:

$$
\begin{aligned}
&C(SCP_d^{Loop}) = N_{loop} \times \sum_{i=1}^{M} C(S_i I_{k_i}) \\
&D(SCP_d^{Loop}) = N_{loop} \times \sum_{i=1}^{M} D(S_i I_{k_i}) \\
&R(SCP_d^{Loop}) = \prod_{i=1}^{M} R(S_i I_{k_i}) \\
&Cor(SCP_d^{Loop}) = \max(Cor(S_1 I_{k_1}),..., \\
&\qquad Cor(S_i I_{k_i}),..., Cor(S_M I_{k_M})) \\
&Bor(SCP_d^{Loop}) = \max(Bor(L^{in}_{S_1 I_{k_1}}),..., Bor(L^{in}_{S_i I_{k_i}}),..., \\
&\qquad Bor(L^{in}_{S_M I_{k_M}}), Bor(L^{out}_{S_M I_{k_M}}), Bor(L_{S_M I_{k_M} \to S_1 I_{k_1}})) \\
&k_i = 1,...,N_i,\ i = 1,...,M,\ d = 1,...,\prod_{i=1}^{M} N_i.
\end{aligned}
\tag{4}
$$

where $L_{S_M I_{k_M} \to S_1 I_{k_1}}$ is the overlay link from $S_M I_{k_M}$ to $S_1 I_{k_1}$ marked by N_{loop} in **Fig. 3-(d)** .

In this section, the conversion and attributes calculation of four basic topologies are discussed. Practically, a complex service composition application can be converted to the simple sequence topology using this mechanism. And then we can calculate QoS and load balance parameters using the sequence topology and solve the MCOSCP problem easily.

2.3.3 Problem Formulation

Suppose there is V candidate SCPs in the MCOSCP problem. The MCOSCP problem is to find a SCP SCP_d which minimizes the load balance metric Z_{SCP_d} while satisfying constraints. The mathematical formulation of MCOSCP problem is presented as follows.

$$Min\ \ Z_{SCP_d} = \alpha \times Cor(SCP_d) + \beta \times Bor(SCP_d) \tag{5}$$

Subject to

$$C(SCP_d) \le C_o \tag{6}$$

$$D(SCP_d) \le D_o \tag{7}$$

$$R(SCP_d) \ge R_o \tag{8}$$

where $d = 1, \ldots, V$. C_o , D_o and R_o are the corresponding objective value of cost, delay and reliability. $C(SCP_d)$, $D(SCP_d)$ and $R(SCP_d)$ are respectively the cost, delay and reliability of SCP_d . $C(SCP_d)$ and $D(SCP_d)$ must be no more than their corresponding objective value. $R(SCP_d)$ must be no less than R_o .

The objective function combines two optimization functions (CPU occupation rate and bandwidth occupation rate) to evaluate the load pressure of overlay nodes and links using this candidate SCP. α and β are the corresponding weights to control the relative significance of *Cor* and *Bor* . This objective function could guarantee that the service selection procedure is able to avoid “hot spots” (SON nodes which are burdened with a heavy load of services) and “hot lines” (SON links which are burdened with heavy traffic) in SON. The MCOSCP problem is a non-linear NP-complete problem [8]. Researchers usually search for approximate solutions for this kind of problems.

3. Niching PSO for Service Composition

3.1 Particle Swarm Optimization (PSO)

PSO is a population based stochastic algorithm proposed by Kennedy and Eberhart [7]. It is one form of swarm intelligence inspired by the behavior of bird flocks. In PSO, a Swarm

composed by m particles flying through M-dimensional search space. When one particle is searching, it updates its position based on its inertia, the best position which it has reached and the best position which all particles in the swarm have reached. In this process, the position of each particle represents a candidate solution to the problem. Additionally, particles have a fitness function to evaluate their best positions and velocities to calculate their inertia. The algorithm is initialized with particles scattering randomly, and searches for the optimal solution by repeating the position update of particles. PSO attempts to balance exploration and exploitation by using personal and social information. Unlike other heuristic algorithms, PSO is easy to implement with less parameters. It takes good effect in solving difficult optimization problems.

At the t[th] iteration ($t = 1,2,...,T_{\max}$), the position of the b[th] particle ($b = 1,2,...,m$) is denoted by $x_b^t = (x_{b1}^t, x_{b2}^t, ..., x_{bi}^t, ..., x_{bM}^t)$, in which x_{bi}^t is the i-dimension position of the b[th] particle. The velocity of b[th] particle is denoted by $v_b^t = (v_{b1}^t, v_{b2}^t, ..., v_{bi}^t, ..., v_{bM}^t)$, in which v_{bi}^t is the i-dimension velocity of the b[th] particle at the t[th] iteration. The best position which the b[th] particle has ever reached is denoted by $p_b^t = (p_{b1}^t, p_{b2}^t, ..., p_{bi}^t, ..., p_{bM}^t)$. The best position which all particles have ever reached is denoted by $p_g^t = (p_{g1}^t, p_{g2}^t, ..., p_{gi}^t, ..., p_{gM}^t)$. The value of fitness function with p_g^t is optimal subject to constraints in the swarm. The i-dimensional position and velocity of b[th] particle is updated as below:

$$\begin{aligned} v_{bi}^{t+1} &= \omega v_{bi}^t + c_1 \xi (p_{bi}^t - x_{bi}^t) + c_2 \eta (p_{gi}^t - x_{bi}^t) \\ x_{bi}^{t+1} &= x_{bi}^t + v_{bi}^{t+1} \end{aligned} \tag{9}$$

where ω is the inertia weight which controls the impact of current velocity to the velocity in the next iteration. c_1 and c_2 are known as learning factors, which together balance the impact of personal and local information. ξ and η are random numbers uniformly distributed between zero and one. The velocity of particles are restricted between $-V_{\max}$ and $V_{\max}$. **Table 1** presents a standard PSO algorithm.

Table 1. A Standard PSO Algorithm

Let

$T_{\max}$ be the max iteration number;

m be the size of the PSO swarm;

$x_b^t = (x_{b1}^t, x_{b2}^t, ..., x_{bi}^t, ..., x_{bM}^t)$ be the b[th] particle's M-dimensional position at the t[th] iteration;

$v_b^t = (v_{b1}^t, v_{b2}^t, ..., v_{bi}^t, ..., v_{bM}^t)$ be the b[th] particle's velocity at the t[th] iteration;

$p_b^t = (p_{b1}^t, p_{b2}^t, ..., p_{bi}^t, ..., p_{bM}^t)$ be the best position which the b[th] particle has ever reached;

$p_g^t = (p_{g1}^t, p_{g2}^t, ..., p_{gi}^t, ..., p_{gM}^t)$ be the best position which all particles have ever reached;

fitness() be the fitness function;

For each particle b **in the swarm:**

Step1: $t = 1$;

Initialize x_b^1 randomly;

Initialize v_b^1 randomly;

Initialize p_b^1, $p_b^1 = x_b^1$;

Initialize p_g^1 to be the one with the best fitness in p_b^1, $b = 1,...,m$;

Step2: Calculate $fitness(x_b^t)$;

Step3: If $fitness(x_b^t)$ is better than $fitness(p_b^t)$, then $p_b^t = x_b^t$;
Step4: Let p_g^t be the one with the best fitness in p_b^t, $b = 1,...,m$;
Step5: Update particle's position and velocity with Eq. 9, $t++$;
Step6: If $t \le T_{\max}$, go to **step2**;
Step7: Output.

3.2 Niching Technique for PSO

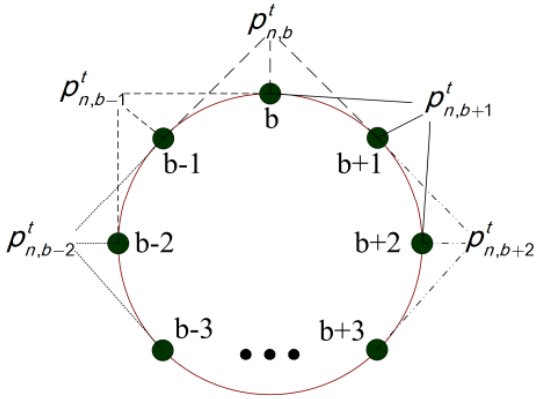


Fig. 4. Niching PSO Structure

Niching technique attempts to find multiple solutions to optimization problems or the global optimal solution in the presence of massive local optimal solutions. Niching technique is oriented in the genetic algorithm (GA) [19] and extensively used in evolution algorithms. Niching technique is classified into two categories: parallel and sequence. A parallel niching method searches for optimal solutions using several niches simultaneously, and could find all optimal solutions in once execution. A sequence niching method searches distinct parts of the solution space using one niche, and finds all optimal solutions iteratively.

There are few PSO algorithms with niching technique proposed recently, including NichePSO [9] and SPSO [10]. NichePSO is the first implementation of niching mechanism in PSO, and adopts a main-swarm and several sub-swarms to balance the exploration and exploitation abilities. SPSO references species genetic algorithm [20] to construct species, and applies normal PSO algorithm to species. Most niching methods of PSO have several tough problems, like reliance on the prior knowledge of the solution space or high computational complexity. Unlike these existing PSO algorithms with niching technique, lbest PSO with ring topology [11] is simple, and needs no prior knowledge. With local memory and a slow communication topology, lbest PSO with ring topology induces stable niching ability. Consequently, it achieves superior performance in locating multiple solutions and finding one global optimal solution in the presence of massive local optimal solutions. Due to its outstanding "cross-trap" capability, we adopt lbest PSO with ring topology (hereinafter Niching PSO) to improve the accuracy of search for optimal solutions in the MCOSCP problem.

In Niching PSO, each particle interacts only with its immediate neighbors. The difference between Niching PSO and standard PSO is the p_g^t of bth particle has changed to the optimal position among p_{b-1}^t, p_b^t, p_{b+1}^t, e.g., bth particle's neighborhood best position $p_{n,b}^t$. The

structure of Niching PSO is illustrated in **Fig. 4**. It uses a "wrap-around" topology, i.e. the first particle is the neighbor of the last particle and vice versa. Eq. 9 could be rewritten as below:

$$\begin{aligned} v_{bi}^{t+1} &= \omega v_{bi}^{t} + c_1 \xi (p_{bi}^{t} - x_{bi}^{t}) + c_2 \eta (p_{n,bi}^{t} - x_{bi}^{t}) \\ x_{bi}^{t+1} &= x_{bi}^{t} + v_{bi}^{t+1} \end{aligned} \tag{10}$$

where $p_{n,bi}^{t}$ is the i-dimensional component of $p_{n,b}^{t}$. The ring topology makes each particle search thoroughly in its local neighborhood before propagating the information throughout the population [11]. This mechanism increases the opportunity of finding the true optimal solution in the MCOSCP solution space.

3.3 Niching PSO for MCOSCP Problem

In this section, we apply Niching PSO to the MCOSCP problem. First we describe the mapping from MCOSCP to Niching PSO domain. Then we present the fitness function selected for MCOSCP problem. Finally, we explain the update of particles, and specify the parameters configuration of the algorithm.

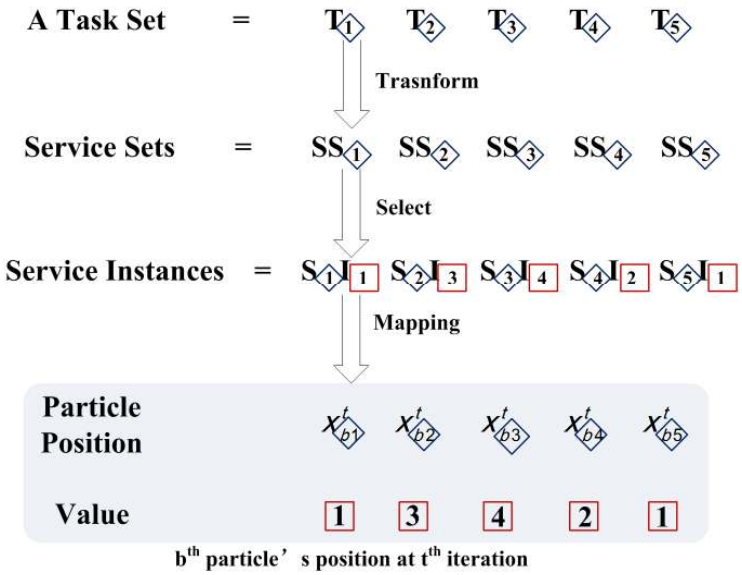


Fig. 5. Mapping MCOSCP to Niching PSO Domain

3.3.1 Mapping MCOSCP to Niching PSO Domain

The most crucial point in order to apply Niching PSO to the MCOSCP problem is mapping MCOSCP to Niching PSO domain. When a service composition request is forwarded to the service directory, it can be denoted by a task set $TS = \{T_1, T_2, ..., T_a, T_M\}$. Then the service directory transforms the task set to service sets $SSs = \{SS_1, SS_2, ..., SS_i, SS_M\}$, as shown in **Fig. 5**. The system uses the Niching PSO algorithm to select optimal service instances for the MCOSCP problem. In this algorithm, the mapping between service instances and a particle in the population is presented in **Fig. 5**. Each index of service set corresponds with a specific dimension number of the particle. Position values of a particle represent instances' indexes, i.e. a possible solution for the problem. Suppose there are m particles in a swarm, possible solutions of the MCOSCP problem are presented by a $m \times M$ position matrix of all particles.

3.3.2 Fitness Function for MCOSCP

The fitness function for the MCOSCP problem not only minimizes the objective function Eq. 5 but also fulfills the constraints Eq. 6-Eq. 8. The fitness function reflects to what degree solutions improve the objective function. In the meantime, it can't violate the constraints Eq. 6-Eq. 8. With Niching PSO, the fitness function guides particles to gather around the optimal solution in the solution space of the MCOSCP problem. The typical fitness function according with these descriptions is a mixture of objective function and penalty function. The objective function relates with Eq. 5 or its transformation. The penalty function is composed of constraints Eq. 6-Eq. 8, and punishes the objective function when a solution violates at least one constraint. According to above descriptions, the fitness function is defined as:

$$Fit(SCP_d) = Z_{SCP_d} + Con(SCP_d) \tag{11}$$

where $Z_{SCP_d} = \alpha \times Cor(SCP_d) + \beta \times Bor(SCP_d)$ is the original objective function representing load factors which will be minimized. The penalty function $Con(SCP_d)$ measures to what extent a solution violates constraints, and equals zero if a solution conforms to all constraints.

$$\begin{aligned} Con(SCP_d) = {} & \lambda_1 \times \max(0,(C(SCP_d) - C_o)/C_o) \\ & + \lambda_2 \times \max(0,(D(SCP_d) - D_o)/D_o) \\ & + \lambda_3 \times \max(0,(R_o - R(SCP_d))/R_o) \end{aligned} \tag{12}$$

where $\max(0,(C(SCP_d) - C_o)/C_o)$ measures the proportion that the cost of SCP_d exceeds C_o when $C(SCP_d)$ violates Eq. 6. Otherwise, it returns zero. Similarly, $\max(0,(D(SCP_d) - D_o)/D_o)$ measures the proportion that the delay of SCP_d exceeds D_o, and $\max(0,(R_o - R(SCP_d))/R_o)$ measures the proportion that R_o exceeds the reliability of SCP_d. λ_1, λ_2, λ_3 are weights to control the importance of $\max(0,(C(SCP_d) - C_o)/C_o)$, $\max(0,(D(SCP_d) - D_o)/D_o)$ and $\max(0,(R_o - R(SCP_d))/R_o)$ in the penalty function which pushes the swarm away from infeasible solutions. According to above statement, it is known that the smaller the value of fitness function is, the better the solution is.

3.3.3 Particles' update

In the t[th] iteration of Niching PSO, the b[th] particle's *i*-dimensional velocity is updated according to the velocity at last iteration, its personal best position p_{bi}^t and neighborhood best position $p_{n,bi}^t$. Then its *i*-dimensional position is updated by the updated velocity. This process is formulated by Eq. 13.

$$\begin{aligned} v_{bi}^{t+1} &= \omega^t v_{bi}^t + c_1^t \xi^t (p_{bi}^t - x_{bi}^t) + c_2^t \eta^t (p_{n,bi}^t - x_{bi}^t) \\ X_{bi}^{t+1} &= \begin{cases} \left\lceil X_{bi}^t + V_{bi}^{t+1} \right\rceil & with\ prob.\ p_1 = (X_{bi}^t + V_{bi}^{t+1}) - \left\lfloor X_{bi}^t + V_{bi}^{t+1} \right\rfloor \\ \left\lfloor X_{bi}^t + V_{bi}^{t+1} \right\rfloor & with\ prob.\ p_2 = 1 - p_1 \end{cases} \end{aligned} \tag{13}$$

where ω^t is the time-variant inertia weight which controls the inheritance from the velocity at last iteration. c_1^t and c_2^t are time-variant learning factors which let the particle move towards personal best and neighborhood best positions. ω^t, c_1^t and c_2^t together with proper values could balance the particle's exploration and exploitation capabilities. ξ^t and η^t are random numbers uniformly distributed between zero and one. They keep the particle moving randomly to escape the local optimum. Due to the value of X_{bi}^{t+1} represents the index of i[th] service instance, it must be an integer. Otherwise, it must be rounded to the nearest smaller integer or to the nearest larger integer randomly such that the mean error is zero. In the next sections, the parameters of Eq. 13 are discussed in detail.

Time-variant inertia weight

The selection of inertia weight is very important for the convergence of the Niching PSO

algorithm. A large inertia weight facilitates exploration to search new areas while a small inertia weight tends to facilitate exploitation to fine-tune the current search area [12]. In PSO, the balance between exploration and exploitation abilities is mainly controlled by the inertia weight [13]. Well, the exploration and exploitation capability of an algorithm is a tradeoff. Different solution spaces need different combinations of exploration and exploitation abilities. In general, particles in the swarm need to explore more to acquire primary knowledge in the initial stage, and need to exploit more to utilize existing information with the iteration increases, especially in the late stage. This can be achieved by using a linearly decreasing inertia weight, as shown in Eq. 14.

$$\omega^t = \omega_{\max} - t(\omega_{\max} - \omega_{\min})/T_{\max} \tag{14}$$

where $\omega_{\max}$ and $\omega_{\min}$ are the maximum and minimum value of inertia weight; $T_{\max}$ is the maximum iteration number. Through empirical studies, Shi and Eberhart [15] have observed that the optimal solution can be improved by varying the value of ω^t from 0.9 ($\omega_{\max}$) at the beginning of the search to 0.4 ($\omega_{\min}$) at the end of the search for most problems.

Asynchronous Time-variant Learning Factors

There are two learning factors in the Niching PSO algorithm. c_1 is the cognitive learning factor which pulls a particle to its best position which it have ever reached. c_2 is the social learning factor which pulls a particle to neighbors' best position. c_1 and c_2 are respectively the weights controlling a particle's memorability of personal best and social best position. If they are small, the particle roams across the search space mainly by inertia. No matter which of them is large, the particle will fly to the local optimum with very large opportunity. So the exploration and exploitation capabilities of a particle could be compromised by tuning the values of inertia weight and learning factors properly. In order to cooperate with linearly decreasing inertia weight, asynchronous time-variant learning factors are proposed [14]. This mechanism enhances global search in the initial stage, and encourages particles to converge to neighborhood optima in the late stage. It can be realized by using decreasing cognitive learning factor c_1^t and increasing social learning factor c_2^t . The values of c_1^t and c_2^t at tth iteration is calculated by Eq. 15.

$$\begin{aligned} c_1^t &= t(c_{1f} - c_{1i})/T_{\max} + c_{1i} \\ c_2^t &= t(c_{2f} - c_{2i})/T_{\max} + c_{2i} \end{aligned} \tag{15}$$

where c_{1i} and c_{1f} are the initial and final values of c_1^t. c_{2i} and c_{2f} are the initial and final values of c_2^t. According to previous research [14], it is effective to adopt settings in Eq. 16 for most situations.

$$c_{1i} = 2.5,\ c_{1f} = 0.5,\ c_{2i} = 0.5,\ c_{2f} = 2.5 \tag{16}$$

Position Range

Because the i-dimensional position value k of the bth particle represents the i^{th} service instance S_iI_k, the range of i-dimensional position x_{bi}^t is the same with that of k in S_iI_k, i.e. $[1, N_i]$. If the i-dimensional position oversteps the range, the candidate solution with it will be invalid. Based on above descriptions, the range of bth particle's i-dimensional position x_{bi}^t is adjusted by Eq. 17.

$$x_{bi}^t = \begin{cases} 1 & if\ x_{bi}^t < 1 \\ N_i & if\ x_{bi}^t > N_i \end{cases} \quad i = 1,...,M \tag{17}$$

Velocity Range

The range of particle's velocity is also important for the algorithm. Small range of velocity

may slow down the search process. Well, large range of velocity may cause particles to fly outside the valid range frequently. In the worst case, particles will oscillate between the minimum and maximum bounds of their positions, i.e. [1, N_i], $i=1,...,M$. The maximum absolute value of velocity needs to be smaller than or equal to the maximum value of corresponding position. Therefore, the value of v_{bi}^t is adjusted by Eq. 18.

$$v_{bi}^t = \begin{cases} -\mathrm{N}_i & if\ v_{bi}^t < -N_i \\ +N_i & if\ v_{bi}^t > +N_i \end{cases} \qquad i=1,...,M \tag{18}$$

Niching PSO Algorithm for MCOSCP Problem

The proposed Niching PSO algorithm for the MCOSCP problem is presented in **Table 2**.

Table 2. Niching PSO Algorithm

Let

$T_{\max}$ be the max iteration number;

m be the size of the PSO swarm;

$x_b^t = (x_{b1}^t, x_{b2}^t, ..., x_{bi}^t, ..., x_{bM}^t)$ be the b[th] particle's M-dimensional position at the t[th] iteration;

$v_b^t = (v_{b1}^t, v_{b2}^t, ..., v_{bi}^t, ..., v_{bM}^t)$ be the b[th] particle's velocity at the t[th] iteration;

$p_b^t = (p_{b1}^t, p_{b2}^t, ..., p_{bi}^t, ..., p_{bM}^t)$ be the best position which b[th] particle has ever reached;

$p_{n,b}^t = (p_{n,b1}^t, p_{n,b2}^t, ..., p_{n,bi}^t, ..., p_{n,bM}^t)$ be the best position among $p_{b-1}^t, p_b^t, p_{b+1}^t$;

$fitness()$ be the fitness function;

For each particle *b* in the swarm:

Step1: $t=1$;

Initialize x_b^1 randomly;

Initialize v_b^1 randomly;

Initialize p_b^1, $p_b^1 = x_b^1$;

Initialize $p_{n,b}^1$ to be the one with the best fitness in $p_{b-1}^1, p_b^1, p_{b+1}^1$;

Step2: Calculate $fitness(x_b^t)$;

Step3: If $fitness(x_b^t)$ is better than $fitness(p_b^t)$, then $p_b^t = x_b^t$;

Step4: Let $p_{n,b}^t$ be the one with the best fitness in $p_{b-1}^t, p_b^t, p_{b+1}^t$;

Step5: Update particle's position and velocity with Eq. 13, $t++$;

Step6: If $t \le T_{\max}$, go to **step2**;

Step7: Output.

4. Performance Analysis

In this section, we evaluate the performance of proposed algorithm on MCOSCP problems. The simulation is composed of two parts: (1) the verification the effectiveness and efficiency of proposed Niching PSO algorithm; (2) the comparison of PSO algorithms with and without the niching technique. We change the number of SSs, the number of service instance, swarm size and iterations for testing the proposed algorithm.

4.1 Simulation Scenarios

In our simulation scenarios, we first use a degree-based AS-level Internet topology generator Inet-3.0 [16] to generate a power-law random graph with 5000 nodes to represent a SON. We then randomly select 10~1000 nodes as SNs, one node as the gateway and another node as the service directory. Once the topology of SON is generated, nodes are connected into a topologically-aware overlay network using Pastry [17]. Service instances are randomly classified into different SSs and distributed to different SNs.

For simulations, we use random QoS parameters with uniform distribution for service instances, including $\text{cost} \sim U[1,5]$, $\text{delay} \sim U[50,700]$ and $\text{reliability} \sim U[90\%,99\%]$. The computing ability of SNs and the CPU ability required by service instances are generated between [5, 100] randomly with uniform distribution. The available bandwidths of overlay links and the bandwidths required by two service instances are generated between [10kb/s, 1000kb/s] randomly. Other common parameters in all test cases are presented below:

- $C_o = 20$ in (6); $D_o = 2500$ in (7); $R_o = 0.65$ in (8);
- $\alpha = \beta = 1$ in (11); $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 100$ in (12);
- $\omega_{\max} = 0.9$, $\omega_{\min} = 0.4$ in (14) as suggested in [15];
- $c_{1i} = 2.5$, $c_{1f} = 0.5$, $c_{2i} = 0.5$, $c_{2f} = 2.5$ in (15) as suggested in [14];

In our simulations, we test each case 10 times and take mean values for final results.

4.2 Simulation Results

We use execution time and accuracy rate (the fitness value of the real optimal SCP versus the fitness value of the optimal SCP which the algorithm finds) as metrics to evaluate the proposed algorithm. The simulation results are categorized into four series. In the first two series, we change the number of SSs (i.e. the number of task sets) and the number of service instances to evaluate the proposed algorithm with increasing swarm size. Then, we test the performance of Niching PSO while the iteration number is increasing linearly. Finally, we compare the performance of Niching PSO algorithm with the performance of standard PSO algorithm when service instances increase in the MCOSCP problem.

4.2.1 Performance with Different Numbers of Service Sets

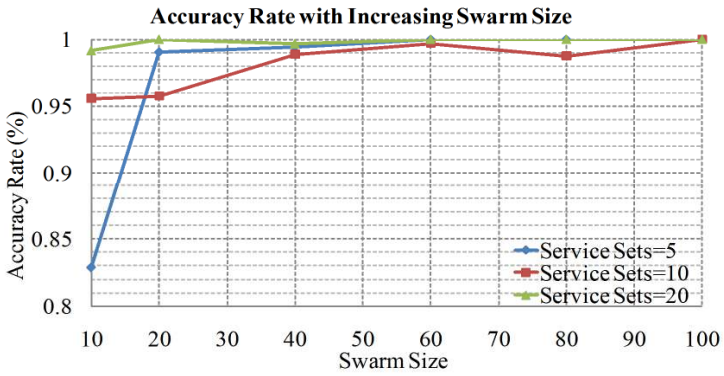


Fig. 6. Accuracy Rate of Niching PSO with Different Numbers of Service Sets

Fig. 6 and **Fig. 7** show the accuracy rate and execution time of Niching PSO when the number of SSs is 5, 10 and 20, respectively. In these tests, the swarm size (number of particles) ranges

from 10 to 100. Well, the number of service instances maintains 3000 at all times. All tests use Niching PSO with 100 iterations. When the swarm size is 10, the accuracy rate with 20 SSs is 20% higher than that with 5 SSs. When swarm size is larger than 20, accuracy rates increase slowly and finally achieve 100%. In the mean time, the number of SSs has little influence over the accuracy rate. The possible reason is service instances in each SS are more than those particles can handle when the numbers of SSs and particles are low. When the swarm size is smaller than 40, the difference of execution time under different numbers of SSs is not obvious. When the swarm size is larger than 60, the execution time with 20 service sets increases in relatively bigger extent. When the number of SSs is 20 and swarm size is 100, it only takes 5.5 seconds to obtain the Pareto solution. The execution time will reduce significantly if obtaining an approximate solution. So the effectiveness and efficiency of the proposed algorithm under different numbers of SSs are proved.

Fig. 7. Execution Time of Niching PSO with Different Numbers of Service Sets

4.2.2 Performance with Different Numbers of Service Instances

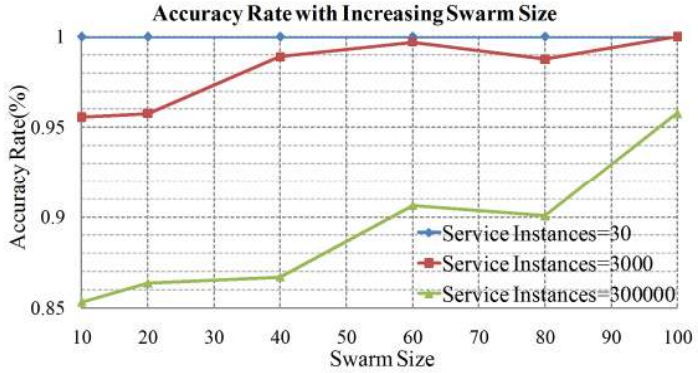


Fig. 8. Accuracy Rate of Niching PSO with Different Numbers of Service Instances

Fig. 8 and **Fig. 9** illustrate the accuracy rate and execution time of Niching PSO when the number of service instances is 30, 3000 and 300000. The number of SSs maintains 10 in all tests. The number of iterations maintains 100 in all tests. As shown in **Fig. 8**, the number of service instances is the key factor affecting the accuracy of Niching PSO. When the swarm size is 10, the accuracy rate with 30 service instances is 5% higher than that with 3000 service

instances, and 17.6% higher than that with 300000 service instances. Gaps among accuracy rates with different numbers of service instances narrow down when the swarm size increases. When the swarm size is 100, the accuracy rate with 30 service instances equals to that with 3000 service instances, and is only 5% higher than that with 300000 service instances. In another aspect, as presented in **Fig. 9**, the execution time is hardly affected by increasing service instances when the swarm size is stationary. When the number of service instances is 300000 and swarm size is 100, it only takes Niching PSO 5.1 seconds to achieve 95.5% approximate value of the Pareto solution. So the effectiveness and efficiency of the proposed algorithm under different numbers of service instances are proved.

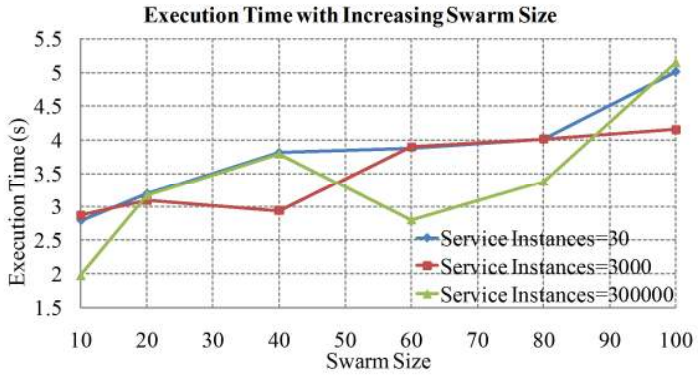


Fig. 9. Accuracy Rate of Niching PSO with Different Numbers of Service Instances

4.2.3 Performance with Increasing Iterations

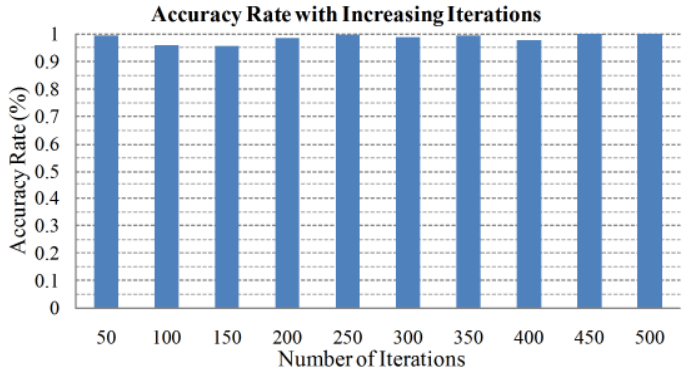


Fig. 10. Accuracy Rate of Niching PSO with Increasing Iterations

Fig. 10 and **Fig. 11** present the accuracy rate and execution time when the number of iterations ranges from 50 to 500. In this process, numbers of SSs, service instances and particles maintain 10, 3000 and 20 respectively. As shown in **Fig. 10**, the accuracy rate improves little when the number of iterations increases. But in this process, the execution time of the proposed algorithm increases almost linearly by a large margin. According to the results drawn from the simulations of this section and the previous two sections, it can be concluded that in order to simultaneously improve the accuracy of niching PSO with little incremental execution time, one must increase the swarm size other than to increase the number of

iterations.

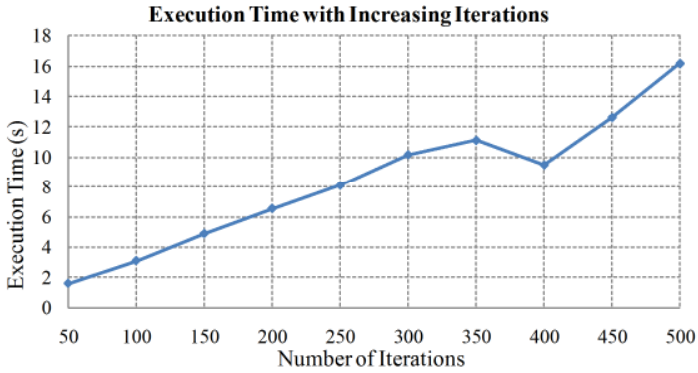


Fig. 11. Execution Time of Niching PSO with Increasing Iterations

4.2.4 Performance Comparison between Niching PSO and PSO

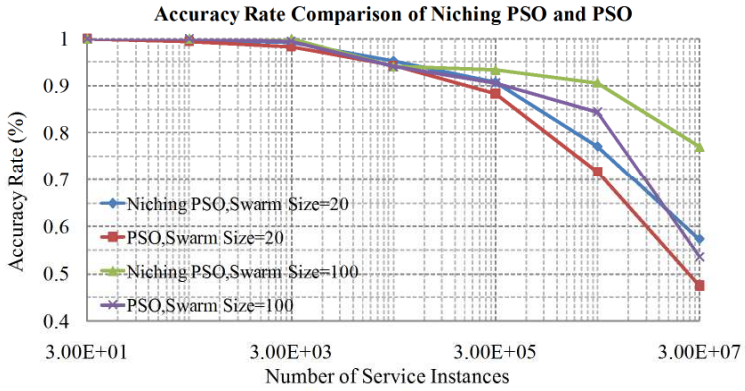


Fig. 12. Accuracy Rate Comparison

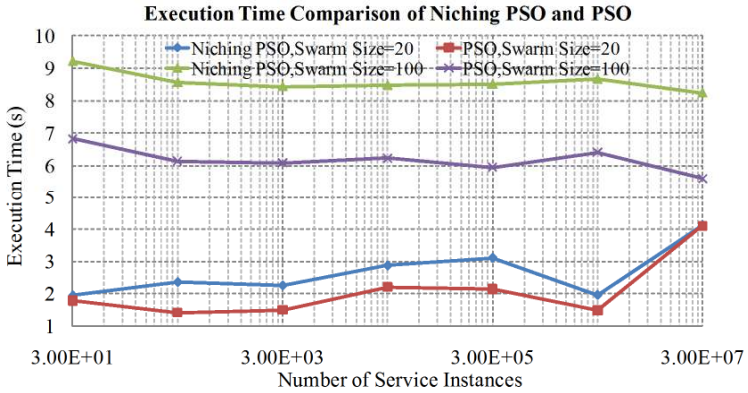


Fig. 13. Execution Time Comparison

In order to compare the performance of the standard PSO algorithm and the Niching PSO algorithm applied for the MCOSCP problem, we implement a Niching PSO algorithm and a standard PSO algorithm on a same MCOSCP problem. In this test scenario, the numbers of SSs and iterations are 10 and 200 respectively. Based on the conclusion of previous sections, the accuracy of the proposed algorithm is mainly influenced by the number of service instances in the MCOSCP problem. So we test two algorithms under cases in which the number of service instances range from 30 to 30000000. First, we use 20 particles in both two algorithms to search for optimal solutions. As shown in **Fig. 12**, when the number of service instances is below 30000, the difference of accuracy rates between Niching PSO and PSO is quite obscure. The gap of accuracy rate between Niching PSO and PSO enlarges with increasing service instances. When the number of service instances is 30000000, the accuracy rate of Niching PSO is 21% higher than that of PSO. In this extreme situation, even the accuracy rate of Niching PSO descends to 57.3%. Then we increase swarm size to 100 to improve the accuracy rate with the number of service instance higher than 30000. When the number of service instances is 30000000, the accuracy rate of Niching PSO is promoted 34%, and the accuracy rate of PSO is promoted 13%. In this situation, the accurate performance of Niching PSO is 43.6% higher than that of PSO. As shown in **Fig. 13**, Niching PSO spend almost 1 more seconds when the swarm size is 20 and 2.5 more seconds when the swarm size is 100 than PSO to achieve better accuracy rate. The accuracy and runtime are a tradeoff in the MCOSCP problem. In service composition, it's quite worth for using the proposed algorithm to improve the accuracy of the system substantially while spending a little more execution time.

5. Related Work

Nowadays, extensive studies have been conducted related to service composition. Papers of this domain mainly refer to two aspects: semantic discussion of service composition and QoS discussion of service composition.

Sirin et al. [1] studied applying hierarchical task network (HTN) planning to automatic web service composition. This paper is the first research proposing an algorithm which translates OWL-S (Ontology Web Language for Services) service descriptions to the HTN planning system SHOP2's (simple hierarchical ordered planner 2) domain.

Kalasapur et al. [4] went further in the semantic research of service composition. They proposed a pervasive information communities organization (PICO) middleware to support service composition on semantic and syntactic layers. The semantic layer represents services' input and output types. The syntactic layer represents services' input and output formats. In service composition, the first step is to check whether there is a path between the input type and the output type of tasks in the semantic layer. If there is a path, the second step is to find the shortest path in the syntactic layer. This paper mainly focuses on the semantic and syntactic service selection in service composition. The QoS-aware service selection is not mentioned in this paper.

In another aspect, there are a few papers addressing QoS in service composition. Zeng et al. [2] proposed local and global algorithms for QoS-aware service composition. This paper uses execution paths and plans to model all possibilities of the composite application. The local algorithm selects the optimal service for each task in the composite application. The global algorithm selects the optimal plan for all paths based on integer programming. Later papers applied this basic approach to diverse scenarios.

Estévez-Ayres et al. [6] proposed off-line and on-line algorithms for real-time service composition. The off-line algorithm computes all possible combination for selecting the optimal service path using a depth first search algorithm. The on-line algorithm improves the previous one by using a pruning technique. These algorithms select optimal services fulfilling QoS constraints with which real-time systems concern. But users' constraints are not mentioned in this paper.

Gu and Nahrstedt [3] presented a decentralized service composition framework, called SpiderNet. SpiderNet maps the function graph into the best qualified ServFlow, which minimizes the utility function and conforms to QoS constraints in the meantime. The bounded composition probing (BCP) algorithm is proposed to probe service instances hop by hop. At last, SpiderNet sorts all qualified ServFlows and then selects the optimal one for execution. However, the service selection is too simple to handle numerous service instances.

Park and Shin [5] proposed three power-aware algorithms for service composition on mobile devices: 1) dynamic QoS control for mobile applications; 2) Reconfiguration of the Service Composition Graph; 3) Service Discovery and Routing. The selection algorithm used in this paper can only deal with linear objective functions.

There are several serious flaws of existing research: 1) users' QoS requirements are not considered; 2) load balance factors are not addressed; 3) the proposed algorithms only aim at specific service composition problems, e.g., problems with one constraint and one objective function or problems with linear constraints and objective functions.

In this paper, we have discussed the service view of load balance in service composition. Previous researches [21][22][23] have addressed the system view of resource allocation. An intelligent market model [21] has been proposed for resource provisioning and allocation in SON. In [22], the inherent selfishness of the peers is taken into account to build a strategic behavior model. Based on this model, a revenue-maximizing auction mechanism has also been presented for the optimization of the path stretch, the link stress, and the aggregate throughput of the network. M. H. Rezvani and M. Analoui [23] proposed distributed algorithms for managing the provisioning of multiple services and allocating the bandwidth to the users.

6. Conclusion

In this paper, we propose a Niching PSO algorithm to fulfill service composition request with QoS constraints and load balance demands. Niching PSO integrates niching technique to promote the "cross-trap" capability of PSO algorithm. Simulation results illustrate that Niching PSO is effective and achieves acceptable levels of efficiency for MCOSCP problem under almost all possible practical circumstances. Moreover, the accuracy rate of Niching PSO is higher than that of PSO, especially with numerous service instances. In the future, we may improve the performance of Niching PSO based on properties of solution space. The development of distributed service composition algorithm is also a recommended future direction.

References

[1] E. Sirin, B. Parsiab, D. Wu, J. Hendler, and D. Nau, "HTN planning for web service composition using SHOP2," *J. Web Semantics*, vol.1, no.4, pp.377–396, Oct.2004. Article (CrossRef Link)

[2] L. Zeng, A.N.B. Benatallah, M. Dumas, J. Kalagnanam and H. Chang, "QoS-Aware middleware for web services composition," *IEEE Trans. Software Eng.*, vol.30, no.5, pp.311-327, May.2004.

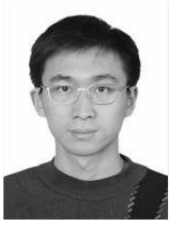
Article (CrossRef Link)
[3] X. Gu and K. Nahrstedt, "Distributed multimedia service composition with statistical QoS Assurances," *IEEE Trans. Multimedia*, vol.8, no.1, pp.141-151, Feb.2006. Article (CrossRef Link)
[4] S. Kalasapur, M. Kumar, and B.A. Shirazi, "Dynamic service composition in pervasive computing," *IEEE Trans. Parallel and Distributed Systems*, vol.18, no.7, pp.907-918, Jul.2007. Article (CrossRef Link)
[5] E. Park and H. Shin, "Reconfigurable service composition and categorization for power-aware mobile computing," *IEEE Trans. Parallel and Distributed Systems*, vol.19 no.11, pp.1553-1564, Nov.2008. Article (CrossRef Link)
[6] I. Estévez-Ayres, P. Basanta-Val, M. García-Valls, J.A. Fisteus, and L. Almeida, "QoS-Aware real-time composition algorithms for service-based applications," *IEEE Trans. Industrial Informatics*, vol.20, no.6, pp.278-288, 2009. Article (CrossRef Link)
[7] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. of IEEE International Conference on Neural Networks*, vol.4, pp.1942–1948, Nov.1995. Article (CrossRef Link)
[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.
[9] R. Brits, A.P. Engelbrecht, and F. van den Bergh, "Locating multiple optima using particle swarm optimization," *Applied Mathematics and Computation*, vol.189, no.2, pp.1859-1883, Jun.2007. Article (CrossRef Link)
[10] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Trans. Evol. Computing*, vol.10, no.4, pp.440–458, Aug.2006. Article (CrossRef Link)
[11] X. Li, "Niching without niching parameters: particle swarm optimization using a ring topology," *IEEE Transaction Evolutionary Computation*, vol.14, no.1, pp.150-169, Feb.2010. Article (CrossRef Link)
[12] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," in *Lecture Notes in Computer Science*, Springer-Verlag, vol.1447, pp.591–600, 1998.
[13] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer", in *Proc. of IEEE International Conference on Evolutionary Computation*, May.1998.
[14] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Trans. Evolutionary Computation*, vol.8, no.3, pp. 240–255, Jun.2004. Article (CrossRef Link)
[15] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proc. of IEEE Int. Congr. Evolutionary Computation*, vol.3, pp.101–106, 1999.
[16] J. Winick and S. Jamin, "Inet 3.0: Internet topology generator," Tech. Rep. UM-CSE-TR-456-02 (http://irl.eecs.umich.edu/jamin/), 2002.
[17] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proc. of International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.
[18] R. Carter and M. Crovella, "Measuring bottleneck link speed in packet switched networks," Tech. Rep. BU-CS-96-006, Comput. Sci. Dept., Boston Univ., 1996.
[19] J.H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, University of Michigan Press, 1975.
[20] J.-P. Li, M. E. Balazs, G. T. Parks, and P. J. Clarkson, "A species conserving genetic algorithm for multimodal function optimization," *Evolutionary Computation* vol.10, no.3, pp. 207–234, 2002. Article (CrossRef Link)
[21] W. Wang and B. Li, "Market-based self-optimization for autonomic service overlay networks," *IEEE Journal on Selected Areas in Communications*, vol.23, no.12, pp.2320-2332, 2005. Article (CrossRef Link)
[22] M. Analoui and M. H. Rezvani, "A framework for resource allocation in multi-service multi-rate overlay networks based on microeconomic theory," *Journal of Network and Systems Management*, vol.19, no.2, pp.178-208, 2011. Article (CrossRef Link)
[23] M. H. Rezvani and M. Analoui, "Strategic behavior modeling of multi-service overlay multicast

networks based on auction mechanism design," *Journal of Parallel and Distributed Computing*, vol.71, no.8, pp.1125-1141, 2011. Article (CrossRef Link)

[24] Z. Duan, Z.-L. Zhang, and Y. T. Hou, "Service overlay networks: SLAs, QoS, and bandwidth provisioning," *ACM Trans. Networking*, vol.11, no.6, pp.870–883, 2003. Article (CrossRef Link)

Jianxin Liao received his Ph.D. degree at University of Electronics Science and Technology of China in 1996. He is currently the dean of Network Intelligence Research Center and the full professor of State Key laboratory of Networking and Switching Technology in Beijing University of Posts and Telecommunications. His current research interests are mobile intelligent network, service network intelligent, networking architectures and protocols, and multimedia communication.

Yang Liu received his B.S. degree at Beijing University of Posts and Telecommunications in 2005 and his M.S. degree at Chinese Academy of Science in 2008. He is currently a Ph.D. candidate at Beijing University of Posts and Telecommunications. His current research interests include service composition, service discovery, particle swarm optimization and clustering method.



Jingyu Wang received his Ph.D. degree at Beijing University of Posts and Telecommunications in 2008. Now he is an assistant professor at Beijing University of Posts and Telecommunications. His research interests include performance evaluation for Internet and overlay network, traffic engineering, image/video coding and multimedia communication over wireless network.

Xiaomin Zhu received his Ph.D. degree at Beijing University of Posts and Telecommunications in 2001. Now he is an associate professor major in Telecommunications and Information Systems at Beijing University of Posts and Telecommunications. His research interests include intelligent networks, next-generation core network and protocol conversion.