Tsinghua Science and Technology

Volume 20 | Issue 6 Article 7

2015

MR-IDPSO: A Novel Algorithm for Large-Scale Dynamic Service Composition

Yanping Zhang
the School of Computer Science and Technology, Key Laboratory of Intelligent Computing and Signal Processing, Ministry of Education, Anhui University, Hefei 230601, China.

Zihui Jing
the School of Computer Science and Technology, Key Laboratory of Intelligent Computing and Signal Processing, Ministry of Education, Anhui University, Hefei 230601, China.

Yiwen Zhang
the School of Computer Science and Technology, Key Laboratory of Intelligent Computing and Signal Processing, Ministry of Education, Anhui University, Hefei 230601, China.

Follow this and additional works at: https://tsinghuauniversitypress.researchcommons.org/tsinghua-science-and-technology

Part of the Computer Sciences Commons, and the Electrical and Computer Engineering Commons

Recommended Citation

Yanping Zhang, Zihui Jing, Yiwen Zhang. MR-IDPSO: A Novel Algorithm for Large-Scale Dynamic Service Composition. Tsinghua Science and Technology 2015, 20(6): 602-612.

This Research Article is brought to you for free and open access by Tsinghua University Press: Journals Publishing. It has been accepted for inclusion in Tsinghua Science and Technology by an authorized editor of Tsinghua University Press: Journals Publishing.

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 07/13 pp602-612
Volume 20, Number 6, December 2015

MR-IDPSO: A Novel Algorithm for Large-Scale Dynamic Service Composition

Yanping Zhang, Zihui Jing, and Yiwen Zhang*

Abstract: In the era of big data, data intensive applications have posed new challenges to the field of service composition. How to select the optimal composited service from thousands of functionally equivalent services but different Quality of Service (QoS) attributes has become a hot research in service computing. As a consequence, in this paper, we propose a novel algorithm MR-IDPSO (MapReduce based on Improved Discrete Particle Swarm Optimization), which makes use of the improved discrete Particle Swarm Optimization (PSO) with the MapReduce to solve large-scale dynamic service composition. Experiments show that our algorithm outperforms the parallel genetic algorithm in terms of solution quality and is efficient for large-scale dynamic service composition. In addition, the experimental results also demonstrate that the performance of MR-IDPSO becomes more better with increasing number of candidate services.

Key words: MapReduce; service composition; Quality of Service (QoS); parallel particle swarm optimization

1 Introduction

Recently, cloud computing has been paid considerable attention as a novel parallel platform. With the advent of cloud computing and Software as a Service (SaaS), it is expected that more and more web services will be offered all over the world[1–4]. Service Composition (SC) aims at reusing and composing existing web services to satisfy complex requirements that cannot be fulfilled by individual services, thus achieving higher-level business value, and becomes very popular when developing service-oriented applications. In the process of SC, a service plan is given as a workflow according to users' request. Service selection is an important step during SC, where a proper service for each task of the plan from a set of candidate services with equivalent function but different QoS attributes is selected[5]. Nevertheless, as the number of candidate services is increasing rapidly, the selection space size of composition is growing dramatically, which increases the need for effective and efficient approaches to solve the problem of large-scale dynamic service composition.

Many researchers have spared no efforts in studying SC in recent years, and proposed many solutions deduced from the perspectives of Particle Swarm Optimization (PSO), Genetic Algorithm (GA), etc.[6–8] Unfortunately, supposing that the number of services continues to grow, these intelligent algorithms may need more iterations and more decision time to find better solutions. From this point of view, the parallelization is always desirable.

With the characteristics of the concurrency of intelligent algorithms, since the early 90th, a series of parallel intelligent algorithms are proposed[9]. However, these algorithms based on general parallel tools—Message Passing Interface (MPI)[10], suffer from a wide range of problems common in data centers, such as inefficient communication, node failure, and unfair load balancing. Thus, it is significant to explore a more

• Yanping Zhang, Zihui Jing, and Yiwen Zhang are with the School of Computer Science and Technology, Key Laboratory of Intelligent Computing and Signal Processing, Ministry of Education, Anhui University, Hefei 230601, China. E-mail: zhangyiwen@ahu.edu.cn.

∗ To whom correspondence should be addressed.

Manuscript received: 2015-03-17; revised: 2015-06-01; accepted: 2015-06-08

suitable solution for performing parallel algorithms in data centers.

The MapReduce programming model[11, 12] has become a promising model for parallel processing in comparison to MPI recently. It was proposed by Google for harnessing large numbers of resources in data centers easily to process data-intensive applications. This model can split a large problem space into small pieces and automatically parallelize the execution of small tasks on the smaller space. MapReduce allows users to benefit from advanced mechanisms due to advanced features of distributed computing which doesn't have to take the difficulty of coordinating the execution of parallel tasks in distributed environments into consideration. Furthermore, the MapReduce technology provides fault-tolerance, load balancing, and data locality.

With the consideration of the above-mentioned problems and situations, we propose a novel algorithm MR-IDPSO for large-scale dynamic service composition. MR-IDPSO is simple, scalable, and robust, on account the fact that it is designed in the MapReduce parallel programming model. The contribution of this paper can be listed as follows:

(1) The proposed algorithm makes use of the Improved Discrete Particle Swarm Optimization (IDPSO) with MapReduce, which achieves the parallelization of IDPSO successfully.

(2) Experiments compared with MR-GA[13], have been conducted on vast amounts of candidate services to show the effectiveness and efficiency of MR-IDPSO for solving large-scale dynamic service composition.

The remainder of the paper is organized as follows: Section 2 describes the model of SC. Section 3 presents the MapReduce programming model and the IDPSO algorithm, as well as our proposed MR-IDPSO algorithm. Section 4 discusses the experimental evaluation. Section 5 presents some related works including QoS-aware service composition and MapReduce. Section 6 concludes the paper with pointers to future work.

2 Model of Service Composition

2.1 QoS-aware service composition

In order to clarify the process of QoS-aware service composition, some basic concepts are listed as below.

- $S=(S^1,\cdots,S^j,\cdots,S^n)$ indicates n abstract services of a composite service.
- $\mathrm{SS}^j=(s_1^j,\cdots,s_h^j,\cdots,s_{m_j}^j)$ denotes the candidate service set for the j-th abstract service S^j of a composite service, which contains m_j candidate services.
- $\mathrm{QoS}=(q_1,\cdots,q_j,\cdots,q_k)$ is the QoS attribute set of services.
- $W=(w_1,\cdots,w_j,\cdots,w_k)$ represents user's preferences, where w_j $(1\leqslant j\leqslant k)$ is the user's preference for the j-th QoS attribute.
- $C=(c_1,\cdots,c_j,\cdots,c_k)$ indicates QoS constraints, where c_j $(1\leqslant j\leqslant k)$ is a QoS constraint over q_j.

In this paper, we only consider the sequential composition model (i.e., service plan), as shown in Fig. 1. Other composition models such as Loop, Parallel, and Conditional can be converted to the sequential composition model through the methods mentioned in Ref. [14]. For each candidate service s_h^j, $\mathrm{QoS}(s_h^j)=(\mathrm{RT}_{s_h^j},\mathrm{RL}_{s_h^j},A_{s_h^j},M_{s_h^j})$ is used to describe its four QoS attributes in aspects of Response Time (RT), Reliability (RL), Availability (A), and Maintainability (M).

Composite Service Executing Paths (CSEP) (i.e., composition solutions) are obtained by selecting one candidate service for each abstract service in the service plan and composing them according to the structure of the plan. Therefore, QoS value of a composite service can be calculated by aggregating the QoS values of corresponding component candidate services. The composite service that not only meets users' functional demands but also has optimal QoS value is regarded as the optimal CSEP.

2.2 QoS aggregation

QoS criteria can be divided into two categories: the positive and the negative. Positive QoS criteria denote better quality with higher values, while negative ones

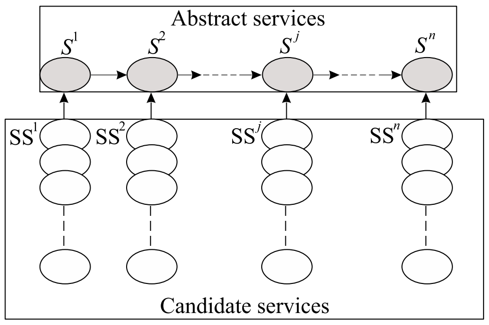

Fig. 1 A predefined workflow model for a composite service.

correspond to lower quality with higher values. In order to avoid inaccurate evaluations due to the various QoS attributes, attribute values must be normalized[15]. Among four QoS attributes considered in this paper, RL, A, and M belong to the positive attributes, which are normalized by Eq. (1). While RT belongs to the negative attributes, which are normalized by Eq. (2).

$$Q'^{j}_{h,j} = Q^{j}_{h,k} / \sum_{h=1}^{n} Q^{j}_{h,k} \tag{1}$$

$$Q'^{j}_{h,j} = 1 - Q^{j}_{h,k} / \sum_{h=1}^{n} Q^{j}_{h,k} \tag{2}$$

where $Q^{j}_{h,k}$ is the k-th QoS attribute of s^{j}_{h}. $\sum_{h=1}^{n} Q^{j}_{h,k}$ is the sum of the k-th QoS attribute of all candidate services in service set SS^{j}. $Q^{j}_{h,k}$ and $Q'^{j}_{h,k}$ refer to the attribute values of a candidate service before and after normalization. It is obvious that QoS attributes of all candidate services are normalized within the interval [0, 1], and higher normalized value indicates better quality.

The optimal selection of SC targets at finding the optimal CSEP from all the possible paths. Suppose that $X = (x^{1}_{h_1}, x^{2}_{h_2}, \cdots, x^{j}_{h_j}, \cdots, x^{n}_{h_n})$ denotes a CSEP (i.e., composition solution), where $x^{j}_{h_j}$ is the index of the candidate service selected from candidate service set SS^{j} for abstract service S^{j}. The QoS model of a CSEP X is $Q_{\text{CSEP}}(X) = (\text{RT}(X), \text{RL}(X), A(X), M(X))$. The method of aggregating the candidates' QoS values to the CS's QoS values is defined as Eq. (3).

$$\begin{cases} \text{RT}(X) = \sum_{j=1}^{n} \text{RT}_{x^{j}_{h_j}}, \\ \text{RL}(X) = \prod_{j=1}^{n} \text{RL}_{x^{j}_{h_j}}, \\ A(X) = \prod_{j=1}^{n} A_{x^{j}_{h_j}}, \\ M(X) = \prod_{j=1}^{n} M_{x^{j}_{h_j}} \end{cases} \tag{3}$$

Then, the fitness value of a CSEP Fitness(X) in this paper is defined as follows:

$$\text{Fitness}(X) = \sum_{k=1}^{4} Q'_{k}(X) \cdot w_k \tag{4}$$

In the expression above, Fitness(X) is the normalized maximum objective function of SC. $\sum_{k=1}^{4} w_k = 1$, where w_k is the weight of k-th QoS attribute of X and represents users' preferences of k-th QoS attribute.

3 Proposed Approach

In view of large-scale solution spaces of SC, a novel parallel approach called MR-IDPSO is proposed. First, we give a basic overview of MapReduce model and IDPSO algorithm. Then, we introduce the customizations of our MR-IDPSO that are tailored to the large-scale service composition problem.

3.1 MapReduce

Cloud computing provides several compelling features such as lower operating cost, higher scalablity, and lower maintenance expenses. Hadoop as an open source cloud-computing platform is widely recognised. MapReduce is a parallel programming model and an associated implementation for processing large data sets of Hadoop[16]. The programming idea of MapReduce is to split vast datasets into many splits. Each split, the size of which is customarily equal to HDFS data block (by default, 64 MB)[17], is assigned to a Map task. All the Map tasks are executed in parallel to produce intermediate results. Then all the intermediate results are integrated on assigned nodes to gain the conclusive results by performing Reduce task. This is how to make a task parallel with this model.

In MapReduce, the problem is formulated as a functional procedure using two core functions: Map function and Reduce function[18].

(1) **Map function**. The Map function regards a key-value pair as input and produces a list of key-value pairs as output. The input key may be of a different type with the output keys, and the input value may be of a different type with the output, which is shown in Eq. (5).

$$\text{Map} :< \text{K1, V1} > \rightarrow \text{list} < \text{K2, V2} > \tag{5}$$

(2) **Reduce function**. The Reduce function regards a key and an associated value list as input and generates a list of new values as output, which is shown in Eq. (6).

$$\text{Reduce} :< \text{K2, list} < \text{V2} >> \rightarrow \text{list} < \text{V3} > \tag{6}$$

The processing of MapReduce framework is shown in Fig. 2.

During the Map period, each split is decomposed into a host of $<$ K1, V1 $>$ pairs to be performed as the input of Mappers. Through the Map function, each $<$ K1, V1 $>$ is transformed into the intermediate data

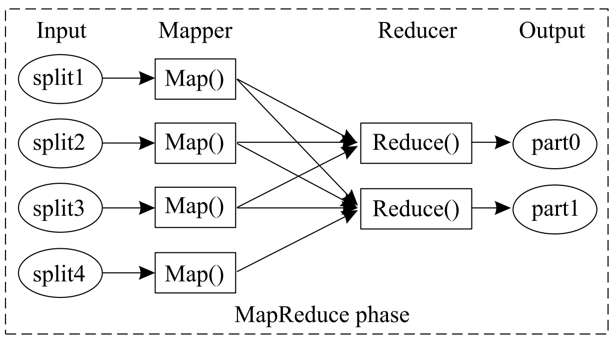

Fig. 2 MapReduce framework.

$< \text{K2, V2} >$ pairs. Then a list of $< \text{K2, V2} >$ pairs from different Mappers are grouped and sorted by key before being sent to reduce tasks.

During the Reduce period, Reducers collect the intermediate results with the same key and then process them to get intended list $< \text{V3} >$. The final results are outputed to HDFS.

3.2 IDPSO

PSO[19] as a parallel intelligent algorithm can be used to solve discrete optimization problems, especially combinatorial optimization problems. It is relatively simple to implement PSO and PSO can quickly converge to a reasonably good solution. Meanwhile, PSO has its shortcomings: (1) may be bogged down in local optimums, (2) generally requires a sufficient number of population size, and (3) demands a sufficient number of iterations. Given that the scale of target problems grows, these defects become more and more severe. For the first shortcoming, we could make some improvements to avoid stagnation. IDPSO aims to solve this shortcoming. However, for the second and the third, parallelization could be a good settlement. Thus, MR-IDPSO is proposed in this paper.

3.2.1 Coding scheme

In order to aggrandize the searching capability of IDPSO, a suitable coding scheme should be designed. In this paper, we use the same coding scheme as Ref. [15]. It is assumed that M particles in a swarm fly in the n-dimensional search space consisting of n circular orbits, where n is also the number of abstract services. The circumference of d-th circular orbit is h_d, and h_d is also the number of candidate services of abstract service S^d. The current position of particle i is denoted as $\boldsymbol{x}_i$, where $\boldsymbol{x}_i = (x_{i,1}, \cdots, x_{i,d}, \cdots, x_{i,n})$ is an n-dimensional vector. $x_{i,d}$ is the position of particle i at d-th circular orbit and a real number within the interval $[0, h_d]$. To evaluate the current position of particle i, it is a need to map the position $\boldsymbol{x}_i$ into the corresponding CSEP^i according to Eq. (7). Then, the fitness value of CSEP^i can be calculated by Eq. (4).

$$\text{fd}(x^t_{i,d}) = \begin{cases} z, (z - 0.5) < x^t_{i,d} < (z + 0.5); \\ \text{rdm}(z, z+1), x^t_{i,d} = (z + 0.5); \\ h_d, x^t_{i,d} \in [0, 0.5) \cup (h_d - 0.5, h_d); \\ \text{rdm}(h_d, 1), x^t_{i,d} = 0.5 \end{cases} \tag{7}$$

From all mentioned above, function fd is utilized to convert $x_{i,d}$ into an index of candidate services in service set SS^d of abstract service S^d. z is an integer variable within the interval $[1, h_d]$, and $\text{rdm}(z, z+1)$ denotes that the value of $\text{fd}(x^t_{i,d})$ equals to z or $z+1$ when $x_{i,d}$ is $z + 0.5$.

3.2.2 Particle movement

The position of a particle represents a potential CSEP and the velocity provides information about direction and changing rate. Particle i determines its movement by its current position $\boldsymbol{x}_i$, the best position it ever found, $p_{i\text{best}}$ (memory), and the global best position achieved by the whole swarm, $p_{g\text{best}}$ (sociability), where $\boldsymbol{x}_i = (x_{i,1}, \cdots, x_{i,d}, \cdots, x_{i,n})$. Furthermore, the movement of a particle is affected by its inertia and other constants. Consequently, particles can search for better position in the regions around $p_{i\text{best}}$ and $p_{g\text{best}}$. The position $\boldsymbol{x}_i = (x_{i,1}, \cdots, x_{i,d}, \cdots, x_{i,n})$, the velocity $\boldsymbol{v}_i = (v_{i,1}, \cdots, v_{i,d}, \cdots, v_{i,n})$, and $p_{i\text{best}}$ are all updated in every iteration step of IDPSO. The best solution $p_{g\text{best}}$ depends on the maximum iteration times (T).

At the beginning, M particles are initialized with a random position. The fitness of all initial positions is evaluated by the fitness function, leading to an initial $p_{g\text{best}}$. The current iteration is denoted as t, then the new position $\boldsymbol{x}^{t+1}_i (t = 1, 2, \cdots, T)$ and the new velocity $\boldsymbol{v}^{t+1}_i$ of particle i are calculated based on the following Eqs. (8) and (9)[15]:

$$v^{t+1}_{i,d} = x^t_{i,d} \otimes (-\text{mod}(2\pi \times |\lambda| \times x^t_{i,d}/h_d, 2\pi)) \otimes \text{mod}(2\pi \times \rho/h_d, 2\pi) \tag{8}$$

$$x^{t+1}_{i,d} = \text{mod}(x^t_{i,d} + v^{t+1}_{i,d}, h_d) \tag{9}$$

where $x^t_{i,d}$ is the position of particle i at d-th circular orbit; $-\text{mod}(2\pi \times |\lambda| \times x^t_{i,d}/h_d, 2\pi)$ is the angle rotating $x^t_{i,d}$ in clockwise direction and $\text{mod}(2\pi \times \rho/h_d, 2\pi)$ is the angle rotating $x^t_{i,d}$ in counter-clockwise direction at d-th circular orbit; $\rho = (c_1\delta p^t_{i\text{best},d} + c_2\eta p^t_{g\text{best},d})$ and $\lambda = -(c_1\delta + c_2\eta)$, where

δ and η are two random values within the interval (0, 1), and c_1 as well as c_2 are calculated by Eqs. (10) and (11), respectively.

$$c_1 = \psi + \cos^2(\psi \times t/T) + \beta \times \mu/\omega \qquad (10)$$

$$c_2 = \psi + \sin^2(\psi \times t/T) + \beta \times \mu/\omega \qquad (11)$$

where ψ is a constant 1.31, β equals 1 or 0, μ is a random value within the interval (0, 1), and $\omega \in [2, 10]$. Interested readers may refer to Ref. [15] for more details about coding scheme and particle movement employed in IDPSO.

3.2.3 Population trimming in IDPSO

In order to prevent premature convergence and enrich the diversity of the population, permutation-based trimming operator[20] is employed in the paper. First, three definitions are given as below.

Definition 1 $\text{dis}_{i,j}$. Boolean distance between particles i and j is defined as follows:

$$\text{dis}_{i,j} = \frac{1}{n}\sum_{d=1}^{n} \text{iff}(\text{fd}(x_{i,d}) \neq \text{fd}(x_{j,d}), 1, 0) \qquad (12)$$

The smaller the value of $\text{dis}_{i,j}$, the smaller the distance between particles i and j.

Definition 2 psdis. Postulate that $\text{dis}_{i,j}$ denotes the Boolean distance between particle i and the global best particle g. Then, psdis is defined as follows:

$$\text{psdis} = \sum_{i=1}^{n} \text{dis}_{i,g}/M \qquad (13)$$

The smaller the value of psdis, the lower the diversity of population.

Definition 3 kma. kma is defined as follows:

$$\text{kma} = \frac{1}{K}\sum_{k=t-K+1}^{t} \text{Fitness}(x_g^k) \qquad (14)$$

where x_g^k is the position of the global best particle g in k-th iteration and the value of K is 10 referring to Ref. [15]. If psdis < 0.35 and the value of kma is unchanging within 15 consecutive iterations, the permutation-based trimming operator will be performed as follows:

Step 1: Delete the redundant particles.

If there are some particles with the same CSEPs, the fitness values of these particles will be equivalent. Then, the system deletes redundant particles and the only one is reserved.

Step 2: Initialize the deleted particles again.

These deleted particles will be initialized again with a random position.

After the two operations above, the diversity of the population is enriched greatly.

3.3 Proposed MR-IDPSO

The nature of PSO makes it an ideal candidate modeled in MapReduce parallel programming for SC. Our MR-IDPSO aims to solve the large-scale SC with parallel IDPSO algorithm and make sufficient use of the simplicity and scalability of MapReduce model. In this paper, the MR-IDPSO consists of three main phases: initialization phase, MapReduce phase, and population trimming phase, as shown in Fig. 3. The procedure of MR-IDPSO illustrates that there is no data exchange between map tasks in the map phase. Reduce is the only way to aggregate the outputs from all map tasks. This restriction ensures the simplicity and reliability of the model.

3.3.1 Initialization phase

In order to reduce the search space of SC and facilitate the process of SC, we introduce Skyline operator[21] as a pre-process before the initialization phase to prune redundant candidate services.

In the initialization phase, an initial particle swarm

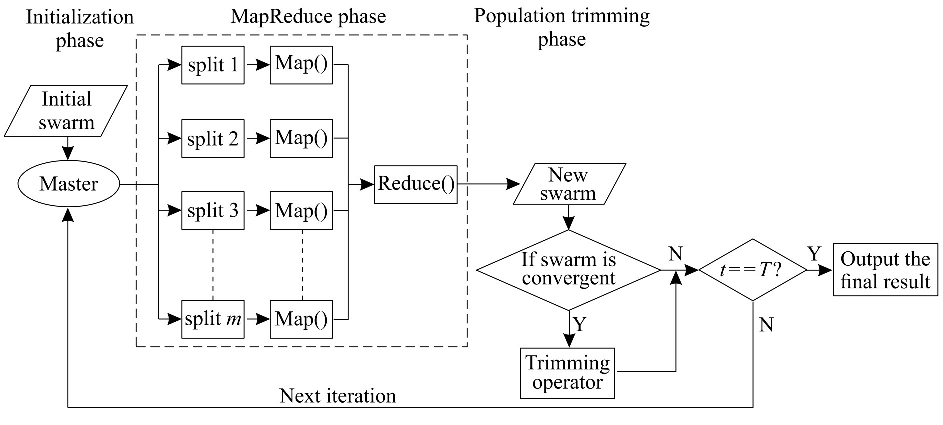

Fig. 3 The procedure of MR-IDPSO.

containing M particles is created. For particle i, a random position $\boldsymbol{x}_i$ is generated using uniform randomization within the given search space. Then, $\boldsymbol{x}_i$ is evaluated by the fitness function. After all initial positions are evaluated, an initial global best position $p_{g\text{best}}$ is generated. The particles are stored in a file on the distributed file system as a $<$ Key, Value $>$ pair structure, where Key is a unique particle ID i and Value is the particle information. The initial stored file is used as input for the first MapReduce job in the MapReduce phase. The representation structure of the $<$ Key, Value $>$ pairs is used in the MR-IDPSO algorithm as shown in Fig. 4. The main components of particle are delimited by semicolon, where the value of id equals to i, the current position $\boldsymbol{x}_i$ is $\boldsymbol{x}_i = (x_{i,1}, \cdots, x_{i,d}, \cdots, x_{i,n})$ and the velocity $\boldsymbol{v}_i$ is $\boldsymbol{v}_i = (v_{i,1}, \cdots, v_{i,d}, \cdots, v_{i,n})$

3.3.2 The MapReduce phase

In the MapReduce phase, an iterative process of MapReduce jobs is performed where each MapReduce job represents an iteration in the particle swarm optimization. The result of each MapReduce job is a fully updated swarm with updated information, which is then used as the input for the next MapReduce job.

(1) Map function

The map function of MR-IDPSO is shown in Algorithm 1. The master splits the input into m splits, where m is the number of map tasks. Thus, the particle swarm is divided into m subpopulations. Each map task updates the particles in its subpopulation independently, while emitting each particle with a constant integer as the key and its updated particle information as the value. In the map task, the $p_{g\text{best}}$ is not updated and the local best particle containing local best position in its subpopulation is selected and recorded. The communication between particles is crucial to find the global best position in the particle swarm. Therefore, the local best particle in subpopulation is also emitted at the end of each map task. Since all the outputs from different map tasks have the equivalent keys, they will be sent to the only reduce task.

(2) Reduce function

The reduce function of MR-IDPSO is shown in Algorithm 2. First, the reduce task combines information from all local best particles to find the global best position in swarm. Then, the reduce task updates the global best position and global best fitness of all particles in swarm with global best information. The reduce task only emits the fully updated particles.

At the end of the MapReduce job, the new particle swarm replaces the previous swarm in the distributed file system, which will be utilized by the next MapReduce job.

I	id; $\boldsymbol{x}_i$; $\boldsymbol{v}_i$; p_{best}; $p_{g\text{best}}$; Fitness($\boldsymbol{x}_i$); Fitness(p_{best}); Fitness($p_{g\text{best}}$)
Key	Value

Fig. 4 Particle representation structure.

Algorithm 1 Map Procedure

```
1: functionMap(Key: MapID, Value: subpopulation)
2: lbestparticle = None
3: for each particle in subpopulation do
4:     particle = Particle(subpopulation)
5:     position^new, velocity^new = PsoMotion(particle)
6:     fitness = Fitness(position^new)
7:     particle.update(position^new, velocity^new, fitness)
8:     if Fitness($p_{\text{best}}^{\text{particle}}$) > Fitness($p_{\text{best}}^{\text{lbestparticle}}$) then
9:         lbestparticle = particle
10:    end if
11:    emit((null, updatedparticle))
12: end for
13: emit((integer, lbestparticle))
14: end function
```

3.3.3 The population trimming phase

In the population trimming phase, the permutation-based trimming operator is performed to maintain diversity of the swarm before sending new particle swarm in the distributed file system to the next MapReduce job. However, it cannot be needed in iterations all the time. If the particle swarm satisfies the condition of performing trimming operator, it represents

Algorithm 2 Reduce Procedure

```
1: function Reduce (Key: Integer, ValList)
2: gbestparticle = None
3: key = 0
4: for each lbestparticle in ValList do
5:     lbestparticle = ParticleLocalbest(ValList)
6:     if Fitness($p_{\text{best}}^{\text{lbestparticle}}$) > Fitness($p_{\text{best}}^{\text{gbestparticle}}$) then
7:         gbestparticle = lbestparticle
8:     end if
9: end for
10: for each particle in ValList do
11:     particle = Particle(ValList)
12:     key = particle.id
13:     particle.update(gbestparticle)
14:     emit((key,particle))
15: end for
16: end function
```

that it still has low evolution and diversity, while the particle swarm executes many times of iterations. Thus, the trimming operator is needed to introduce outside excellent particles and enhance the quality and diversity of swarm. After performing the trimming operator, the newly updated particle swarm is sent to the next MapReduce job, replacing the previous swarm in the distributed file system.

Otherwise, there is no need to consider the trimming operator. The new particle swarm emitted from last MapReduce job in the distributed file system is sent to the next MapReduce job directly.

4 Experimental and Discussions

In this section, we compare the optimization quality of our MR-IDPSO with that of MR-GA[13] to verify the feasibility and effectiveness of our algorithm for large-scale dynamic service composition problem. Considering the runtime of per iteration can be used to evaluate the performance of MR-IDPSO, in this paper we also test the time of per iteration with the increasing number of candidate services, mappers or processors.

4.1 Environments and date set

4.1.1 Environments

We ran our parallel experiments on a Hadoop cluster. The Hadoop cluster consists of only 5 nodes equipped with 2 GB of RAM and Intel Xeon E3-1230 v3 8 cores (3.3 GHz each). One node is set as the master node. The remaining nodes are set as data nodes. A MapReduce job with 4 map tasks and 1 reduce task is initiated. Besides, we adopted Hadoop version 2.4 for the MapReduce framework, and Java runtime 1.6 for the system implementation.

4.1.2 Data set

It is assumed that there is a model of sequential service composition process consisting of seven abstract services. Unless noted, 10 000 candidate services are generated for each abstract service to form a sufficiently large dataset. In our experiments, four typical QoS properties are taken into consideration. QoS values of candidate services in abstract services are randomly generated within certain ranges refering to Table 1.

For all experiments, the size of population is fixed to 1000. In MR-GA, the probabilities of crossover and mutation are set as 0.8 and 0.1, respectively. Due to the randomness of intelligent algorithms, each case of the experiments runs 20 times.

Table 1 The QoS attribute settings for generating candidate services.

QoS attribute	Value range
Response time	(0, 2000)
Availability	[0.8, 1.0)
Reliability	[0.5, 1.0)
Maintainability	[0.2, 1.0)

4.2 Performance of the MR-IDPSO compared with MR-GA

4.2.1 Comparison on the number of iteration times

This experiment aims to graphically take on the evolutionary performance when solving large-scale service composition problem with the increasing number of iteration times. The results are depicted in Fig. 5a, where the abscissa stands for the iteration times and the vertical axis denotes the average fitness over 20 runs. As we can see in Fig. 5a, the average fitness of MR-IDPSO as well as MR-GA, rises slowly with the increasing number of iteration times. Figure 5b shows the average, the worst, and the best fitness values of solutions after all cases run 20 times.

From Fig. 5, it can be seen that MR-IDPSO outperforms MR-GA regardless of exploration ability for promising solutions and exploitation ability for optimal solution. Meanwhile, these results indicate that the permutation-based trimming operator introduced into MR-IDPSO enables to prevent premature convergence and enhance the particles' searching

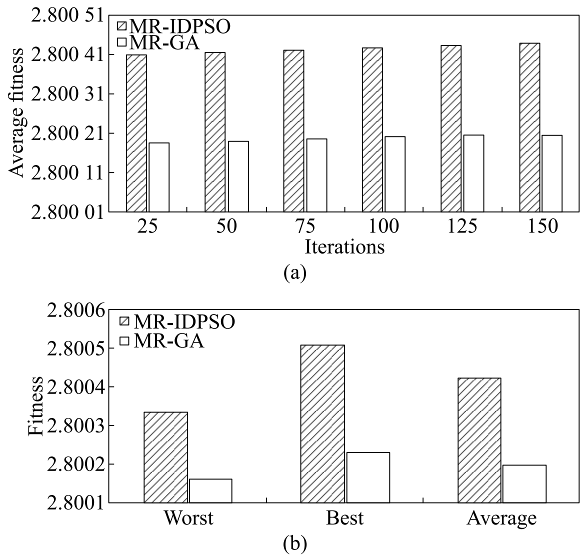

Fig. 5 Performance of MR-IDPSO and MR-GA with varying itration times.

ability for service composition. Thus, MR-IDPSO finally converges to much better results than MR-GA.

4.2.2 Comparison on the number of candidate services

This experiment is to verify the effectiveness and efficiency of MR-IDPSO for large-scale service composition problem. In all cases, iteration times are set as 100. We report the average fitness of solutions over 20 runs. The experimental results over different number of candidate services are shown in Fig. 6.

It can be seen from Fig. 6a that, MR-IDPSO algorithm is superior to MR-GA on the ability of finding optimal solutions. The average fitness of MR-IDPSO generally rises with the increase of candidate services. Nevertheless, this tendency is also not kept always. For example, one can see that the average fitness drops slightly, after the number of candidate service increases to 5000.

Figure 6b shows the runtime per iteration with the increase of candidate services. From Fig. 6b, it is obvious that our MR-IDPSO is more efficient than MR-GA in all the cases. When the number of candidate services is 10 000, the runtime per iteration of MR-GA is 77.77 s, while MR-IDPSO is 24.67 s. The growth speed of the time per iteration of MR-IDPSO is significantly slower than that of MR-GA. Due to the skyline operator introduced into MR-IDPSO, some redundant services can be pruned to facilitate the process of service composition quickly. Moreover, MR-GA requires ranking all chromosomes between reduce and map phase, which results in additional cost. However, it is needless in our MR-IDPSO. Thus, our algorithm outperforms MR-GA in terms of efficiency. The disparity widens as the number of candidate services increases. Therefore, it is concluded that the MR-IDPSO proposed in this paper has better effectiveness and more efficiency for large-scale service composition problem.

4.3 Scalability of MR-IDPSO

The scalability of MR-IDPSO is essential in terms of the runtime. In this experiment, the iteration times are set as 100. As shown in Fig. 7a, at the beginning, the time per iteration decreases slightly as more and more mappers are added. Given that population size is not large enough in this paper, this tendency seems not very apparent. However, it can be concluded that adding more mappers decreases the time per iteration when the population size is large enough. But the time gradually increases after 4 mappers. The reason is that the communication time increases with mappers increasing.

We ran the MR-IDPSO with 4, 8, 16, 24, and 32 processors. In each case, we report the average execution time of 100 iterations. Figure 7b shows the time overhead of MR-IDPSO. As shown in Fig. 7b, the runtime decreases faster for 4 processors and 8 processors than that of 32 processors at the end. The runtime of 4 processors is 98.7 s, while 32 processors only consume 24.6 s for the same problems. The overhead of the Hadoop framework drops nearly linear with the increase of processors. It is indicated that in

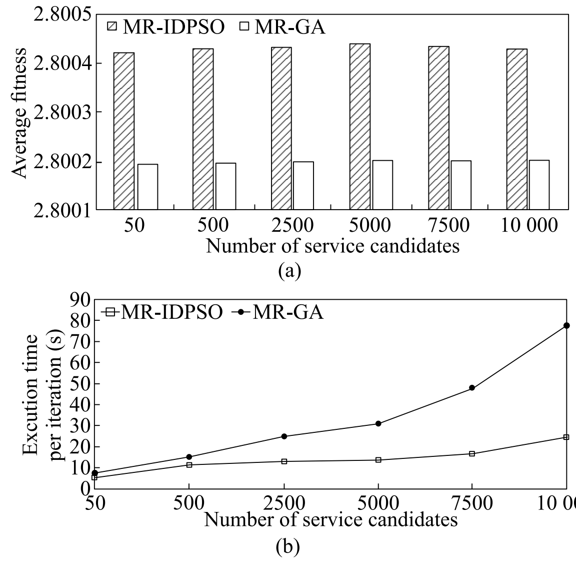

Fig. 6 Performance of MR-IDPSO and MR-GA with varying candidate services.

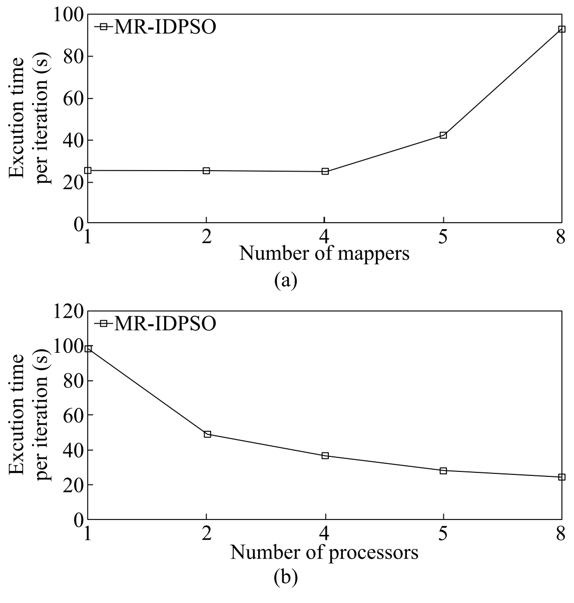

Fig. 7 Scalability of MR-IDPSO.

order to improve the searching efficiency further, more processing power is needed. Based on the experiments above, it can be concluded that the MapReduce model is extremely useful to process large-scale service compositon problems.

5 Related Work

In this section, we briefly discuss some of the works related to QoS-aware service composition problem. The literatures on the application of MapReduce are also investigated in the following studies.

Wang et al.[22] proposed an efficient and effective QoS-aware service selection approach. It employed cloud model to compute the QoS uncertainty for pruning redundant services while extracting reliable services. The authors of Ref. [21] introduced skyline operator as a pre-process before service composition to address the problem of Cloud-based web service composition. Alrifai et al.[23] proposed an algorithm whose idea is to decompose the global SLAs constraints into local ones. Doing so, they showed that we can quickly obtain good approximations for the service selection problem. However, as services proliferate, these methods may fail to facilitate large-scale service composition problem quickly. Beran et al.[24] described two parallel algorithms for the service selection problem. The first one is a master-slave parallelization of a genetic algorithm. Their second proposition is the parallelization of an $A*$ like algorithm for service selection. Tao et al.[25] proposed a parallel chaos optimization algorithm to solve service composition efficiently. In Refs. [26, 27], two parallel algorithms for service composition are proposed.

Many parallel models like fine grained[28], coarse grained[29], and distributed models[30] were proposed for implementing parallel intelligent algorithms. However, most of the parallel models are based on MPI technology that is not the best choice for parallelization because of the weakness of handling the failure of nodes.

As the basic large data processing framework, MapReduce provides simple approaches for massive data. The authors of Ref. [31] incorporated the MapReduce model to parallelize particle swarm optimization, and applied it to performing data-intensive tasks. Aljarah and Ludwig[32] proved successfully that the Glowworm Swarm Optimization (GSO) can be parallelized easily with the MapReduce technology. In Ref. [33], the authors made use of skyline with MapReduce to select skyline services in parallel from massive web services belonging to one task, which is optimal service selection for single task instead of service composition optimal selection for multitask. However, in the work above, particle swarm optimization algorithm in combination of MapReduce for service composition problem has not been researched. Therefore, in this paper, we design a novel parallel PSO algorithm incorporating the MapReduce to parallelize PSO for addressing large-scale service composition problem.

6 Conclusions

With the proliferation of available services in cloud computing, service composition problem needs to be addressed efficiently. In this paper, a novel parallel particle swarm optimization algorithm, MR-IDPSO, for addressing large-scale service composition problem, is designed. With the consideration of large irregular solution spaces of service composition, skyline operator is introduced as a pre-process before service composition to decrease the range of service selection. In order to improve the searching efficiency further, IDPSO is parallelized with the MapReduce framework. In addition, the permutation-based trimming operator is employed to overcome premature convergence of particles and enhance the solution quality. The experimental results demonstrate that our algorithm can be more effective and efficient than MR-GA for large-scale service composition problem.

In the future, we will improve the efficiency and effectiveness of our algorithm for larger scale service composition problem further. Moreover, how to apply our approach to other combinatorial optimization problems will be also studied in the future.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (No. 61175046), the Natural Science Foundation of Anhui Province of China (No. 1408085MF132).

References

[1] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, Cloud computing — The business perspective, *Decision Support Systems*, vol. 51, no. 1, pp. 176–189, 2011.

[2] Z. Chen, W. Y. Dong, H. Li, P. Zhang, X. M. Chen, and J. W. Cao, Collaborative network security in multi-tenant data center for cloud computing, *Tsinghua Science and Technology*, vol. 19, no. 1, pp. 82–94, 2014.

[3] A. Klein, I. Fuyuki, and S. Honiden, SanGA: A self-adaptive network-aware approach to service composition, *IEEE Transactions on Services Computing*, vol. 7, no. 3, pp. 452–464, 2014.

[4] A. Jula, E. Sundararajan, and Z. Othman, Cloud computing service composition: A systematic literature review, *Expert Systems with Applications*, vol. 41, no. 8, pp. 3809–3824, 2014.

[5] J. F. Chen, H. H. Wang, D. Towey, C. Y. Mao, R. B. Huang, and Y. Z. Zhan, Worst-input mutation approach to web services vulnerability testing based on SOAP messages, *Tsinghua Science and Technology*, vol. 19, no. 5, pp. 429–441, 2014.

[6] X. Zhao, B. Song, P. Huang, Z. Wen, J. Weng, and Y. Fan, An improved discrete immune optimization algorithm based on PSO for QoS-driven web service composition, *Applied Soft Computing*, vol. 12, no. 8, pp. 2208–2216, 2012.

[7] H. Liang, Y. H. Du, and S. J. Li, An improved genetic algorithm for service selection under temporal constraints in cloud computing, in *Web Information Systems Engineering-WISE 2013*, Springer, 2013, pp. 309–318.

[8] Y. W. Zhang, G. M. Cui, Y. Wang, and S. Zhao, An optimization algorithm for services composition based on improved FOA, *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 90–99, 2015.

[9] T. G. Crainic and M. Toulouse, Parallel meta-heuristics, in *Handbook of Metaheuristics*. Springer, 2010, pp. 497–541.

[10] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*. MIT Press, Cambridge, MA, USA, 1995.

[11] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] J. B. Zhang, D. Xiang, T. R. Li, and Y. Pan, M2M: A simple Matlab-to-MapReduce translator for cloud computing, *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 1–9, 2013.

[13] N. Khalid, A. Fadzil, and M. Manaf, Adapting MapReduce framework for genetic algorithm with large population, in *IEEE Conference on Systems, Process & Control (ICSPC)*, IEEE, 2013, pp. 36–41.

[14] H. Zheng, J. Yang, W. Zhao, and A. Bouguettaya, QoS analysis for web service compositions based on probabilistic QoS, in *Service-Oriented Computing*. Springer, 2011, pp. 47–61.

[15] T. Wen, G. J. Sheng, Q. Guo, and Y. Q. Li, Web service composition based on modified particle swarm optimization, (in Chinese), *Chinese Journal of Computers*, vol. 36, no. 5, pp. 1031–1046, 2013.

[16] Y. X. Zhao, J. Wu, and C. Liu, Dache: A data aware caching for big-data applications using the MapReduce framework, *Tsinghua Science and Technology*, vol. 19, no. 1, pp. 39–51, 2014.

[17] Apache Hadoop, *http://en.wikipedia.org/wiki/Apache Hadoop*, 2015.

[18] S. Babu, Towards automatic optimization of MapReduce programs, in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ACM, 2010, pp. 137–142.

[19] J. Kennedy, Particle swarm optimization, in *Encyclopedia of Machine Learning*. Springer, 2010, pp. 760–766.

[20] F. Tao, D. Zhao, H. Yefa, and Z. Zhou, Correlation-aware resource service composition and optimal-selection in manufacturing grid, *European Journal of Operational Research*, vol. 201, no. 1, pp. 129–143, 2010.

[21] S. Wang, Z. Zheng, Q. Sun, H. Zou, and F. Yang, Cloud model for service selection, in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2011, pp. 666–671.

[22] S. Wang, Q. Sun, H. Zou, and F. Yang, Particle swarm optimization with skyline operator for fast cloud-based web service composition, *Mobile Networks and Applications*, vol. 18, no. 1, pp. 116–121, 2013.

[23] M. Alrifai, T. Risse, and W. Nejdl, A hybrid approach for efficient web service composition with end-to-end QoS constraints, *ACM Transactions on the Web*, vol. 6, no. 2, pp. 7:1–7:31, 2012.

[24] P. P. Beran, E. Vinek, E. Schikuta, and M. Leitner, An adaptive heuristic approach to service selection problems in dynamic distributed systems, in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, Washington, DC, USA: IEEE Computer Society, 2012, pp. 66–75.

[25] F. Tao, Y. LaiLi, L. Xu, and L. Zhang, FC-PACO-RM: A parallel method for service composition optimal-selection in cloud manufacturing system, *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2023–2033, 2013.

[26] J. Pathak, S. Basu, R. Lutz, and V. Honavar, Parallel web service composition in moscoe: A choreography-based approach, in *IEEE 4th European Conference on Web Services*, IEEE, 2006, pp. 3–12.

[27] P. Bartalos and M. Bielikov'a, Semantic web service composition framework based on parallel processing, in *IEEE Conference on Commerce and Enterprise Computing*, IEEE, 2009, pp. 495–498.

[28] A. Brunetti, A fast fine-grained genetic algorithm for spectrum fitting: An application to X-ray spectra, *Computer Physics Communications*, vol. 184, no. 3, pp. 573–578, 2013.

[29] Y. Chen and X. Wei, Study on coarse-grained parallel genetic algorithm, *Advanced Materials Research*, vol. 532, pp. 1654–1658, 2012.

[30] D. Lim, Y. S. Ong, Y. Jin, B. Sendhoff, and B. S. Lee, Efficient hierarchical parallel genetic algorithms using grid computing, *Future Generation Computer Systems*, vol. 23, no. 4, pp. 658–670, 2007.

[31] A. W. McNabb, C. K. Monson, and K. D. Seppi, Parallel pso using MapReduce, in *IEEE Congress on Evolutionary Computation*, IEEE, 2007, pp. 7–14.

[32] I. Aljarah and S. A. Ludwig, A MapReduce based glowworm swarm optimization approach for multimodal functions, in *IEEE Symposium on Swarm Intelligence (SIS)*, IEEE, 2013, pp. 22–31.

[33] L. Pan, L. Chen, and J. Wu, Skyline web service selection with MapReduce, in *IEEE International Conference on Computer Science and Service System (CSSS)*, IEEE, 2011, pp. 739–743.

Yanping Zhang received her PhD degree in computer science from Anhui University in 2003. She is currently a professor in Department of Computer Science and Technology, Anhui University. Her primary research interests include computational intelligence, quotient space theory, artificial neural networks, and machine learning.

Zihui Jing is a master student in the School of Computer Science and Technology, Anhui University. She received her BEng degree in computer science from Anhui University in 2013. Her current research interests include service computing and cloud computing.

Yiwen Zhang received the master degree in computer software and theory in 2006 and the PhD degree in management science and engineering in 2013 both from the Hefei University of Technology. He is an associate professor in the School of Computer Science and Technology at Anhui University. His research interests include service computing, cloud computing, and e-commerce.