Deploying Data-Intensive Service Composition with a Negative Selection Algorithm

Shuiguang Deng, College of Computer Science and Technology, Zhejiang University, Hangzhou, China
Longtao Huang, College of Computer Science and Technology, Zhejiang University, Hangzhou, China
Ying Li, College of Computer Science and Technology, Zhejiang University, Hangzhou, China
Jianwei Yin, College of Computer Science and Technology, Zhejiang University, Hangzhou, China

ABSTRACT

With the development of information technology, data on the Internet is growing even faster than Moore’s Law. At the age of big data, more and more services are created to deal with big data, which are called data-intensive services. In most cases, multiple data-intensive services are assembled into a service composition to meet complicated requirements. Since the big-data transmission, which is occurred among component services as well as between a service and a data center, has great influence on the overall performance of a composition, deploying those services cannot be considered independently. This paper proposes an optimal deployment method based on a negative selection algorithm for a data-intensive service composition to reduce the cost of the data transmission. When making a deployment schedule, it considers not only the cost of data transmission among component services, but also the load balance of data centers where component services are deployed. It models the deployment problem as a combination optimization problem and extends a negative selection algorithm to get an optimal deployment plan. A series of experiments are carried out to evaluate the performance of the proposed method using different settings as well as to compare with other methods. The results show that the method outperforms others for the problem of data-intensive service composition deployment.

Keywords: Big Data, Data-Intensive, Deployment, Information Technology, Service Composition

INTRODUCTION

With the development of Service Oriented Computing (Papazoglou et al., 2008), web service technology has attracted much attention from industry and academia in recent years and achieved significant success. Service composition is one of the most important issues in SOC, and has become a main technology enabler for
delivering cloud solutions with its ability of enabling the interoperation of heterogeneous systems, reuse of distributed functions in an unprecedented scale and creating value-added business applications (Jiang et al., 2012; Deng et al., 2013).

The explosion of data and information has been recognized in recent years and people have stepped into the age of big data. Generated data is growing too fast to store and handle as before. Emerging cloud-based infrastructures for storage have been widely accepted as the next-generation solution to address the data proliferation and the reliance on data (Kolodner et al., 2012). Data-intensive services are now being developed and utilized in more and more fields (Deng et al., 2013). The emergence of cloud computing and data-intensive services brings both huge changes and new challenges for Service Oriented Computing technology. Traditionally, service providers usually deployed services on their own local infrastructures. Due to the explosion of data and economic benefits, nowadays service providers prefer to deploy their services in cloud. Generally, these services are deployed in the data centers of cloud providers (e.g. Amazon, Google). For traditional service compositions, the time cost of data transmission among component services is negligible compared to the execute time of component services. So most research focuses more attention on how to effectively make service composition, but rarely considers the problem of deploying service composition. However, the data transmission among components in data-intensive service compositions is incredibly huge. If the component services are deployed independently and casually, the latency of the whole composition would be incogitable. Hence, it is essential to make an optimal deployment strategy for data-intensive service compositions for reducing latency of user requests.

At present, limited work has focused on the problem of deploying service compositions. The work in Kang et al. (2012) may be a little similar to ours, but they aimed at deploying multiple correlated services in cloud. However, service composition is much more complicated because the component services in compositions may be organized in different structures (Rao & Su, 2005). And there are complex dependency relationships (data/logic dependency) among component services. Dependency relationships, especially data dependency relationships, have a great influence on the overall performance of data-intensive service compositions. In this paper, we mainly consider the following factors to tackle the issue of deploying data-intensive service compositions:

1. **Mass Data Transmission:** For a data-intensive service composition, data transmission across different data centers is inevitable. Besides, the amount of transmitted data is quite huge. Hence, it requires taking consideration of the network bandwidths between datacenters and the transmitted data size comprehensively to make an optimal deployment schedule and reduce the overall latency of the composition;

2. **Dependency Relationships:** The component services in a data-intensive service composition may have multiple logic/data dependency relationships with each other. The components having dependency relationships should be deployed as close as possible. It can reduce the cost of data transmission;

3. **Load Balance:** To make an optimal deployment schedule, we should also consider maintaining a relatively load balancing of data centers.

In this study, we transformed the problem of data-intensive service composition deployment to a combination optimize problem. Then we proposed a novel method based on the negative selection algorithm to make optimal deployment schedules for data-intensive service compositions. To evaluate the effectiveness of our proposed method, serial simulation experiments were conducted. The comprehensive experimental results show that our proposed mechanism can provide good performance and
scalability. The contributions of this paper focus on the following three aspects:

- Firstly, we address the new research problem of deploying data-intensive service compositions. And we also provide a formalized definition for the problem;
- Secondly, we introduce negative selection algorithm to solve the problem of data-intensive service compositions deployment. And we make an extension for the negative selection algorithm to make optimal deployment schedules for data-intensive service compositions;
- Finally, we simulate, compare and analyze the necessity, feasibility and performance of our approach for the problem of deploying data-intensive service compositions.

The rest of this paper is organized as follows. Section 2 introduces the negative selection algorithm and reviews some related work. Section 3 presents the problem definition and the formal models used in the following parts. Section 4 describes the main operations and algorithms of our proposal. Section 5 shows our evaluation experiment and the results. Section 6 concludes this study and discusses future directions.

MOTIVATING SCENARIO

We use the China Mobile Limited in China as an example to illustrate the problem of data-intensive service composition deployment. China Mobile Limited is the largest mobile telecommunications company in China. It currently jumps on the cloud computing bandwagon through building up several large data centers across China to store large-scale data and provide various mobile Internet services. Some of the services are too complex to be implemented by only one data center. Such services, in general, are implemented by composing some component services offered by different data centers according to logics; and also they need utilize data from different data centers. Due to the tremendous cost occurred from the interactions among services and the transmission of big-data between services and data centers, it is a critical issue to deploy such a service composition correctly in order to ensure it running efficiently.

Consider such a scenario shown in Figure 1 in which China Mobile Limited wants to deploy a service composition $S$ which is composed of five component services namely $s_a$ to $s_e$ and needs to deal with massive data from four different data centers namely $dc_1$ to $dc_4$. To deploy the service composition $S$ is to deploy each component service to a proper data center. However, which data center a component service should be deployed on is concerned with not only where the data the service will deal with, but also other factors such as the cost of data transmission and the cost of service interaction.

As the data centers are located in different provinces across China, the network distances vary between these data centers. Furthermore, there would be massive data transmission among the component services and the sizes of data transmission between component services are also different. Consider service $s_c$, it processes $d_3$ from data center $dc_2$ and $d_2$ from service $s_b$, and output $d_4$ to service $s_e$. So when we decide which data center it should be deployed on, we should take all the cost of data transmission into consideration, which is influenced by not only the size of data to be transmitted but also the distance of transmission. Moreover, the deployment of $s_c$ should not be considered separately due to its interactions between $s_b$ and $s_e$. That is to say, all the interacted services should be considered together when determining their deployments. Hence, it’s essential to make a global optimal deployment schedule for a service composition in order to ensure it running effectively.

BACKGROUND AND RELATED WORK

In this paper, we present a negative selection based optimization method to make optimal
deployment schedules for data-intensive service compositions. This is based on the negative selection algorithm. In this section, we will first introduce some basic knowledge on the negative selection algorithm to set the scene for our extensions. Then we will review some related work on service deployment.

**Overview of NSA**

The negative selection algorithm (NSA) belongs to the field of Artificial Immune Systems (AIS) (Cao et al., 2007). AIS can be defined as a class of computationally intelligent systems inspired by theoretical immunology and observed immune functions, principles and models, which have been successfully utilized to solve problems in many fields (Dasgupta et al., 2011). The algorithms typically exploit the human immune system (HIS)’s characteristics of learning and memory to solve a problem. There are four major AIS algorithms which have been constantly researched and have acquired wide popularity: (1) clonal selection algorithm (De Castro & Von Zuben, 2000); (2) negative selection algorithm; (3) artificial immune network (De Castro & Timmis, 2002); and (4) dendritic cell algorithm (Greensmith et al., 2006).

For most optimization algorithms, such as genetic algorithm (Goldberg, 1989), particle swarm optimization (Kennedy & Eberhart, 1995), differential evolution (Storn & Price, 1997) and so on, the basic idea is to obtain the optimal solutions recursively. Compared to the above optimization approaches, the negative selection algorithm takes a completely different methodology compared to other regular iteration-based optimization algorithms (Laurentys et al., 2010). NSA is inspired by the positive and negative selection processes that occur during the maturation of T cells in the thymus called T cell tolerance. The basic operations of NSA are introduced in Figure 2: (1) initialize the genes warehouse, (2) obtain the immature immune cells or antibodies through gene recombination; (3) remove the self ingredients in the immature immune cells through negative selection principle, and then mature antibodies are obtained; (4) Evaluate the obtained antibodies according a fitness function; (5) If the mature antibodies
recognize antigen well, the consistence of alleles will be self-adaptively increased in the genes warehouse management. Otherwise, it will be decreased. Repeat the steps 2)-5) and more and more solutions with bad patterns will be deleted from the search space with the progression of the algorithm, effectively accelerating the convergence process for the optimal solution.

RELATED WORK ON SERVICE DEPLOYMENT

In distributed computing systems, research on data and service placement has been widely studied (Cope et al., 2009; Hardavellas et al., 2009; Zhang et al., 2010). However, most aim at solving single service or single server deployment problem. Rarely does work focus on reducing the data transmission between data centers on the Internet. The following works may have some similarities with ours.

Yuan et al. (2010) examined the unique features of scientific cloud workflows. Data movement between data centers has the tendency to take a long time, since data centers are spread around the Internet with limited bandwidths. In their work, they try to place the application data based on their dependencies in order to reduce the data movement between data centers. They proposed a clustering data placement strategy that can automatically allocate application data among data centers based on the dependencies. In our work, we only consider reducing the times of data movements. The time cost of
data transmission may not reduce with shorter times of data movements. Rather, it is affected by other factors such as the size of transmitted data, the network bandwidth. Furthermore, we aim at deploying services while Yuan’s work made data placement.

Kang et al. (2012) pointed out the issue of multiservice co-deployment in cloud computing. They aimed at making optimal deployment for multiple correlated services. The multiservice co-deployment problem was modeled formally as an integer-programming problem, which minimizes the latency of user requests. While their work is similar to ours, multiple correlated services are not service compositions. The structure of a service composition is more complicated. There are multiple kinds of logic/data dependency relationships among component services. So it is a much more challenging research problem of deploying data-intensive service compositions. In addition, they did not consider the load balance problem when deploying services.

**PROBLEM DEFINITION AND MODELING**

In order to discuss the issue of deploying data-intensive service compositions properly, we need some prerequisites. Therefore, in this section we formally define the key concepts used in data-intensive service composition and the deployment problem:

**Definition 1 (Data-intensive Service Composition):** We use a directed graph to represent a data-intensive service composition and it can be modeled as a 3-tuple \( dsc = \langle S, L, D \rangle \) where:

1. \( S \) is the set of component services in the composition. Besides the actual component services, we model two types of virtual services: a) The set of initial services \( I \) will provide initial data for the service composition, where the number of initial service is determined by the number of initial data sources. Consider the example in Figure 1, three initial services are generated to provide \( d_0 \) for \( s_a \), \( d_3 \) for \( s_c \), and \( d_6 \) for \( s_e \); b) The service \( e \) is an end service which ends the service composition and returns the required data;
2. \( L = \{ ldep_{ij} \mid s_i, s_j \in S \} \) is the set of logic dependencies between services;
3. \( D = \{ ddep_{ij} \mid s_i, s_j \in S \} \) is the set of data dependencies between services;

**Definition 2 (Logic Dependency):** A Logic Dependency defines the order of execution among services. At present, we mainly consider the following types of logic dependencies:

- **Sequence:** A component service is executed after the completion of another component service. For example, \( s_5 \) and \( s_6 \) are executed in sequence as shown in Figure 3;
- **AND-split:** A logic node that makes a single process into multiple processes which can be executed in parallel, thus allowing component services to be executed simultaneously. For example, \( s_5 \) and \( s_7 \) can be executed in parallel after the AND node;
- **AND-join:** A logic node where multiple parallel processes converge;
- **XOR-split:** A logic node that makes a single process into multiple processes and only one can be selected to be executed. Note that OR-split is a similar logic but differs in that the number of processes to be selected ranges from 0 to the total number of processes;
- **XOR-join:** A node where two or more alternative branches come together without synchronization;
- **Loop:** The component services may be executed several times repeatedly. Here we suppose the number of iterated times to be fixed;
Definition 3 (Data Dependency): A Data Dependency \( \text{ddep}_{ij} = \langle s_i, s_j, \text{data}_{ij} \rangle \) defined between two component services indicates that the output data/message \( \text{data}_{ij} \) from \( s_i \) is used as an input of \( s_j \). The relation of date dependency is a basic characteristic of data-intensive service compositions.

Figure 3 is an example of a data-intensive service composition in which we use solid directed edges to represent logic dependencies as opposed to the dotted edge to represent data dependencies. As an example for data dependency, \( s_4 \) data-depends on \( s_3 \). The services \( s_0 \), \( s_1 \) and \( s_2 \) are initial services and the service \( s \) is the end service:

Definition 4 (Data Centers): The set of data centers where services can be deployed is modeled as \( \text{DC} = \langle \text{sc}_i, \text{st}_i \rangle \), where \( \text{sc}_i \) is the storage capacity of \( \text{dc}_i \), and \( \text{st}_i \) is the load threshold fraction of \( \text{dc}_i \). The network bandwidth between each pair of data centers is modeled as a matrix \( \text{BW} \), where \( \text{BW} \)'s element \( \text{BW}_{ij} \) represents the bandwidth between \( \text{dc}_i \) and \( \text{dc}_j \). And there are two characteristics of \( \text{BW} \): 1) \( \text{BW}_{ij} \) = 0, when \( i = j \); 2) \( \text{BW}_{ij} = \text{BW}_{ji} \):

Definition 5 (Deployment Schedule): Given a data-intensive service composition \( \text{dsc} \) and a set of data centers \( \text{DC} \), a deployment schedule is modeled as:

\[
\text{ds} = \bigcup_{i=1,2,...,|\text{dsc}|} \{ s_i \rightarrow \text{dc}_j \ | \ \text{dc}_j \in \text{DC} \}
\]

That is, for each component service in \( \text{dsc} \), find a data center in \( \text{DC} \) to deploy on. Different
component services may be deployed on one data center:

**Definition 6 (Latency):** The latency of a data-intensive service composition is the overall time from accepting a request to return the results:

\[
\text{Latency} = T_{\text{data}} + T_{\text{exec}} + T_c
\]  

(1)

where \(T_{\text{data}}\) is the cost time of data transmission among component services. If the component services which have data dependencies are deployed in the same data center, then there is no time cost of data transmission between them. \(T_{\text{exec}}\) is the cost time of service executions. \(T_c\) is other time cost such as requesting, responding, making/stopping connections, etc. For a data-intensive service composition, the size of data transmission among component services is huge, so \(T_{\text{exec}}\) and \(T_c\) is negligible comparing to \(T_{\text{data}}\). Then:

\[
\text{Latency} \approx T_{\text{data}}
\]  

(2)

**Definition 7 (Overload Rate):** Given a data center \(dc\), the overload rate of \(dc\) is:

\[
R_{\text{overload}}(dc) = \begin{cases} 
0 & \text{usage} < dc.st \\
\frac{\text{usage}}{dc.sc} & \text{otherwise} 
\end{cases}
\]  

(3)

where \(\text{usage}\) is the overall data transmitted to \(dc\). The overall overload rate of the set of data centers in \(DC\) is:

\[
R_{\text{overload}}(DC) = \sum_{dc_i \in DC} R_{\text{overload}}(dc_i)
\]  

(4)

**Definition 8 (Problem Definition):** Given a data-intensive service composition \(dsc\) and a set of data centers \(DC\), each component service is deployed in one data center in \(DC\) and it can be executed only if the data it requires is transmitted to the data center where it is deployed. Hence, the goal is to find the optimal deployment schedule taking consideration of the cost of data transmission among component services and the load balancing of data centers. Then the problem is in fact transformed to a combination optimize problem. The objective is to obtain the best deployment schedule with the lowest \(\text{Latency}(dsc)\) and the lowest \(R_{\text{overload}}(DC)\).

**MAIN OPERATIONS AND ALGORITHMS**

**Overview**

In the negative selection algorithm the problems to be solved are regarded as antigens, and then the optimization problem of data-intensive service composition deployment corresponds to an antigen. The deployment solutions correspond to antibodies. While antibodies are composed of genes, the chosen data center for a component service corresponds to a gene in an antibody. The component services in a data-intensive service composition correspond to alleles. In order to measure the quality of a solution, the fitness of the corresponding antibody is calculated. In this study, we take the common genes of antibodies with high fitness as heuristic knowledge to accelerate the speed of convergence.

Considering the negative selection algorithm, we use the basic steps of the original negative selection algorithm in section 3. To enhance the performance of the algorithm for data-intensive service composition deployment, we also make some extensions for NSA. The details of our approach are illustrated in the following parts.
Initialization Operation

In general, negative selection algorithm works by generating a population of antibodies (solution vectors), each representing a possible solution to the data-intensive service composition deployment problem. Hence, an integer array is used to represent the chosen data centers and maps a solution vector to a specific data-intensive service composition. For example, an antibody vector \( (2, 3, 1, 2) \) means there are four component services in the data-intensive service compositions, where the first component service is deployed in the data center 2, the second component service is deployed in the data center 3, and so on.

In the initialization operation, we regard the chosen data centers for data-intensive service composition deployment problem as the gene segments, initialize all the possible values of the gene segments, and put them into the genes warehouse. As a result, all the composite arrays for the genes warehouse form the total solution space for the to-be-solved problem. Then, the antibody population size is set, which determines how many solution vectors are generated for each iteration. And the maximum iteration number is set to determine when to terminate the algorithm.

Besides, the other two variables are initialized: the Consistence Matrix and the SELF set:

- **Consistence matrix**: Consistence matrix is used to evaluate whether the chosen data center for a component service is appropriate or not. The consistence matrix \( C \) contains \( p \times q \) elements, where \( p \) is the number of component services in the data-intensive service composition and \( q \) is the number of the data centers for deployment. All the elements of \( C \) are initialized as 1. The consistence matrix will be updated in the gene warehouse update operation;

- **SELF set**: The SELF set is used to record those genes with poor quality, that is, the unsuitable data centers for specific component services would be put into the SELF set. It is initialized an empty set and dynamically updated in the gene warehouse update operation.

Gene Recombination Operation

In this step, the goal is to generate a population of antibodies. The population size is set in the initialization operation. Each antibody consists of \( p \) genes, where \( p \) is the number of component services in the data-intensive service composition. For an antibody \( v \), the \( i \)-th gene is selected according to the function \( \text{selectGen}(v, i) \). The \( j \)-th data center is selected for the \( i \)-th allele (component service) based on a dynamic probability:

\[
\text{probability}(i, j) = \frac{C_{i,j} \cdot \text{localfitness}(i, j)}{\sum_{k=1}^{q} C_{i,k} \cdot \text{localfitness}(i, k)}
\]

where \( C_{i,j} \) is the element in the consistence matrix, \( q \) is the number of the data centers for deployment. The function \( \text{localfitness}(i, j) \) is used to evaluate whether it is suitable to deploy the \( i \)-th component service in the \( j \)-th data center. The calculation of \( \text{localfitness}(i, j) \) is:

\[
\text{localfitness}(i, j) = \frac{\text{dc}_{j,sc} - \text{data}_i}{\text{dc}_{j,sc}}
\]

where \( \text{data}_i \) is the data size that the \( i \)-th component service needs as inputs. And \( \text{dc}_{j,sc} \) is the storage capacity of the \( j \)-th data center. Hence, the more remaining capacity the \( j \)-th data center has after the \( i \)-th component service is deployed, the higher the local fitness \( \text{localfitness}(i, j) \).

In reality, some services can only be deployed in the fix data centers. Then the select probability of these services for the specific data centers is set 1.

In order to accelerate the converging speed of the algorithm, we also make an extension for NSA in this step. In original NSA, all the antibodies are generated based on the function \( \text{selectGen} \). In our method, we keep the optimal
antibody in the last iteration, and put it into the population of the current iteration. Then the others are generated as above. The benefit is that the gene advantages of the ancestors are kept. As a result, the speed of convergence will be accelerated a lot.

**Negative Selection Operation**

The negative selection operation is the foundation of negative selection algorithm and can effectively reduce the searching space. It is responsible for removing the antibodies with poor quality. For each newly obtained antibody through the genes recombination operation, all the genes of it will be matched with the SELF set. If the number of the antibody’s genes in the SELF set is larger than a threshold $\beta$, the antibody is considered unfit and is removed from the population. Also, if an antibody is not valid, such as the required data size is out of the storage capacity of the data center, it will be removed.

**Gene Evaluation Operation**

In the gene evaluation operation, the fitness of each antibody is calculated to evaluate its quality. For the problem of deploying data-intensive service compositions, we mainly consider two factors which contribute to the fitness of a deployment schedule: the latency of the composition and the load balancing of data centers. The fitness function for one deployment schedule $v$ is calculated as Equation (7) based on Equation (2) and Equation (4):

$$fitness(v) = \omega_1 \cdot \mathcal{R} \cdot Latency^{-1} + \omega_2 \cdot R_{overload}^{-1}$$  \hspace{1cm} (7)

where $\omega_1$ and $\omega_2$ are the weights for the two variables. The weights can be set according to different preferences. In this paper, we set $\omega_1=\omega_2=0.5$. And $\mathcal{R}$ is the adjustment normalizing factor to balance the two variables.

**Gene Warehouse Update Operation**

The aim of this step is to obtain better antibodies in the gene recombination operation of next iteration.

We first divide the antibodies into two groups based on their fitness. The antibodies with higher fitness than average will be put into the *good* group, while the others are put into the *poor* group. For the antibodies in the *good* group, we update the consistence matrix according to their genes. Given an antibody $v$, if the $i$-th gene chooses the $j$-th data center, then the element $C_{i,j}$ will be updated according to:

$$C_{i,j}' = \rho \cdot C_{i,j} + \frac{fitness(v)}{fitness_{\max}}$$  \hspace{1cm} (8)

where $\rho$ is the attenuation coefficient which closely relates to the convergence of the algorithm. And $fitness_{\max}$ is the maximum value of the fitness in the population.

For the antibodies in the *poor* group, we will put the poor genes into the SELF set. If the *local fitness* of the gene is less than the self-threshold $\alpha$, it will be considered as a poor gene and be put into the SELF.

After the gene warehouse update operation, the consistency of alleles in excellent antibodies is increased, while the consistency of alleles in relatively poor antibodies is decreased. This results in better selection in the gene recombination operation of next iteration.

**EXPERIMENT AND EVALUATION**

In this section, we conduct a series of experiments to evaluate and validate our proposed approach. The experiments are aimed at answering the following questions:

1. How does the parameters $\alpha, \beta$, and $\rho$ impact the performance of our approach?
2. How does our approach compare with other metaheuristic algorithm-based methods when solving the issue of scheduling data-intensive service compositions?

Our experiments are developed by Java and the execution environment is: Intel Core2 P7370 2.0GHZ with 4GB RAM, Windows 7, and jdk1.6.0. In the following, we provide the details of our experiments.

**Evaluation of Parameters Impact**

To evaluate the impact of three important parameters: $\alpha$, $\beta$, and $\rho$, experiments are conducted with a moderate scale of a data-intensive service composition scheduling problem including 10 component services and 10 servers for scheduling. The data-dependency relationships and logic-dependency relationships among components are generated randomly. Also, the store capacity and overload threshold of each data center and the bandwidth between them are generated randomly. For our approach, the antibody population size $S=10$, maximal iteration number is 500 and algorithms are independently run 50 times for unprejudiced statistical results.

The parameter $\alpha$ is used to determine whether an allele is inserted into the SELF set. The number of elements in the SELF set will increase when $\alpha$ becomes larger. The parameter $\beta$ is used to determine which genes are deleted from antibody population. An antibody should be removed if the antibody contains more than $\beta$ genes in the SELF. Attenuation coefficient $\rho$ is used to measure the inertia of the consistence matrix. To evaluate the impact of the three parameters, we set the value of them according to Table 1. Figure 4 shows the average best fitness in 50 independent runs per iteration.

Figure 4(a) shows that the performance of algorithm with $\alpha=0.1$ performs better than those with $\alpha=0.3$ and $\alpha=0.5$. When $\alpha$ becomes larger, the negative selection process will abandon more antibodies and the non-self/negative space decreases dramatically. As a result, the convergence of algorithm is accelerated which leads to a lower quality of the solution because the larger $\alpha$ is, the more possible it is to delete the optimal solution and arrive at a local optima.

Figure 4(b) shows that the parameter $\beta$ hardly has clear influence on the solution accuracy and computing efficiency. In fact, the eliminated antibody genes cannot be too much for the relative small antibody population. Therefore, $\beta$ affects the negative selection strategy slightly. As a result, it has little effects on the convergence and efficiency.

Figure 4(c) shows that $\rho=1$ out-performs the other two. When $\rho$ is relatively small, the inertia of the consistence matrix will decrease causing the algorithm to converge rapidly and its performance to become unsteady. The selection probabilities of excellent genes are increased, leading to a rapid converging speed of antibody population. But, if the excellent genes haven’t been selected, the density of the gene will attenuate naturally, and the probability for it being selected will become even smaller for the next time.

As a result, our method can perform better with the combination of parameters: $\alpha=0.1$, $\beta=1$ and $\rho=1$.

<table>
<thead>
<tr>
<th>Test</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>0.1,0.3,0.5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Test2</td>
<td>0.1</td>
<td>1,5,10</td>
<td>1</td>
</tr>
<tr>
<td>Test3</td>
<td>0.1</td>
<td>1</td>
<td>0.5,0.9,1</td>
</tr>
</tbody>
</table>
COMPARISON WITH OTHER ALGORITHMS

Since little work has focused on the problem of scheduling data-intensive service composition, we can hardly find existing methods for such problem to compare with ours. Because NSA is a fundamental metaheuristic algorithm, we try to use other metaheuristic algorithms to solve this problem and compare with our method. Clonal Selection Immune algorithm (CSA) and Genetic Algorithm (GA) are chosen for comparisons. The parameters for each algorithm are set as shown in Table 2.

First, we compare the algorithms in Table 2 with different iteration number. The data of

Table 2. Algorithm parameters

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSA</td>
<td>$\alpha = 0.1$, $\beta = 1$ and $\rho = 1$</td>
</tr>
<tr>
<td>CSA</td>
<td>Mutate rate = 0.25, Cloning numbers =10</td>
</tr>
<tr>
<td>GA</td>
<td>Crossover rate = 0.8, Mutate rate = 0.25</td>
</tr>
</tbody>
</table>
service compositions and servers is the same as the pre-section. We set the iteration number 500, 1000, and 5000. The performance comparisons are show in Figure 5, 6, and 7.

From the comparison results, we can conclude that the convergence speed of NSA is much faster than that of CSA and GA.

In order to evaluate the performance and scalability of the algorithms for the problem of scheduling data-intensive service compositions, various scales of service composition instances and data centers are generated randomly for performance comparison. Three groups of data are generated as shown in Table 3.

Figure 8, 9, and 10 shows that the NSA has an outstanding performance on convergence speed and solution quality with all the three datasets. Especially in Figure 10 when the number of component services and servers becomes very large, our approach out-performs the other two by a large margin. From this we
can conclude that our approach performs better even with large scale datasets.

CONCLUSION AND FUTURE WORK

Table 3. Various scales of datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Component Services</th>
<th>Number of Servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Dataset2</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>Dataset3</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>

Figure 7. Comparison of the algorithms with different iteration number - iteration number = 5000

Figure 8. Comparison of the algorithms with different scales of datasets - Dataset1
In this paper, we investigated the data-intensive service composition deployment problem. We transformed this problem to a combination optimize problem. Then we introduced the negative selection algorithm to solve the problem, and a heuristic extension for the algorithm is given to accelerate the speed of convergence. A series of experiments are conducted to evaluate the necessity, feasibility and performance of our approach. The experimental results prove the effectiveness of our approach. In the future, we plan to study the deployment problem in more complex situations. The constraints on networks, I/O and other factors will be considered. And we will also attempt to make deployment schedules for hybrid service compositions with computation-intensive services, data-intensive services, and I/O-intensive services.
ACKNOWLEDGMENT

This research was supported in part by the National High-Tech Research and Development Plan of China under Grant 2011BAD21B02 and the National Natural Science Foundation of China under Grant No. 61170033.

REFERENCES


Shuiguang Deng is with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. He received the Doctor’s degree in computer science from Zhejiang University, Hangzhou, China, in 2007. He is currently an associate professor in Zhejiang University. His research interests include service computing, cloud computing and middleware techniques.

Longtao Huang is with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. He received the Bachelor’s degree in software engineering from Zhejiang University, Hangzhou, China, in 2010. Now study the PhD of computer science and technology in Zhejiang University from 2010. His research interests include service computing and cloud computing.

Ying Li is with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. He received the Doctor’s degree in software technology from Zhejiang University, Hangzhou, China, in 2000. He is currently an associate professor in Zhejiang University. His research interests include software architecture, software automation, compiling technology, and middleware techniques.

Jianwei Yin is with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. He received the Doctor’s degree in computer science from Zhejiang University, Hangzhou, China, in 2001. He is currently a professor in Zhejiang University. His research interests include distributed network middleware, software architecture and information integration.