Parallel Optimization for Data-intensive Service Composition

Shuiguang Deng¹, Longtao Huang¹, Bin Wu¹, Lirong Xiong²

¹College of Computer Science and Technology, Zhejiang University, China
²College of computer Science, Zhejiang University of Technology, China
{dengsg, hlt218, wubin}@zju.edu.cn, lilybear@zjut.edu.cn

Abstract

The age of Big Data has inspired the appearance and application of data-intensive Web service. In most cases, multiple data-intensive services are assembled into a service composition to meet complicated requirements. As the number of Data-intensive Web services on the Internet is increasing rapidly and dramatically, traditional central service composition approaches have come to a performance bottleneck. This paper proposes a method for automatic data-intensive service composition, which can be executed in parallel. Firstly, the problem of automatic service composition is defined by combing the approaches of State Space Search and Planning Graph. Then, a heuristic algorithm is proposed to compose data-intensive Web service automatically. After that, more details are given to present the parallel optimization for the composition algorithm. A series of experiments show that the proposed parallel optimization method improves the efficiency of automatic Web service composition to a great extent.

Keywords: Data-intensive service, Service composition, Parallel optimization.

1 Introduction

Service-oriented computing is an emerging paradigm for developing software by reusing distributed components called services [1]. As a key technology in Service-oriented computing, service composition has attracted much from both industry and academia areas and achieved significant success. This success is due to the capability of services to integrate data and enterprise applications [2][13].

Nowadays, with the rapid development and wide application of the new generation of information technology, such as Cloud Computing, Internet of Thing and Mobile Internet, the information society has stepped into the age of Big Data. Big data has inspired the appearance and application of data-intensive service, which has grown to be an important service type in the Internet and has attracted much attention immediately. Due to its heavily dependence on big data processing, data-intensive service has brought much challenges to traditional service models and service coordination methods.

In most cases, multiple data-intensive services are assembled into a service composition to meet complicated requirements. As the number of data-intensive Web services on the Internet is increasing dramatically, it’s already beyond the human ability to compose services manually from large-scale candidate services. Thus, the automatic Web service composition aiming at generating a composite service to meet the user request has become an important technique to reuse existing resources and accelerate the development of Web applications [3]. Many researchers consider service composition as an AI planning problem. The planning graph method has been used to model the problem and reduce the search space greatly [4-6]. Sirina et al propose an approach by using SHOP2, which is a successful HTN planning system [7]. This approach is suitable for automatic service composition because the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in OWL-S process ontology. Based on the HTN approach, many variants emerge: Chen et al propose a model of combining a Markov decision process model and HTN planning to address services composition [8]. Besides SHOP2, there are also other planners. Seog-Chan Oh et al present a novel solution named as BF* that claims to adopt the competitive A* as a search algorithm [9]. However, the BF* algorithm is not really an A* algorithm but just a heuristic search algorithm. Furthermore, backtrack is not considered in the BF* algorithm, thus it does not search all possible ways. In other words, it is exactly a heuristic construction algorithm, which constructs the plan by the heuristic function. This algorithm could not guarantee that the found plan is the best (shortest) path.

However, the above approaches are almost based on central mechanism. With the number of data-intensive services growing too much, it may lead to a performance bottleneck due to the explosion of the planning and searching space. In order to improve the efficiency of automatic Web service composition with large-scale data-intensive Web services, we make an extension of our previous work in [10] by combining the state space and the planning graph, and optimizing it with a decentralized way to make the composition process executed in parallel. This approach takes the advantages of the state space approach and the planning graph approach to construct a parallel composition framework based on multi-agents.
The parallel composition framework consists of a central agent and several planning agents. The central agent is the interface between the whole composition system and the outside application system. It distributes the planning task to all the planning agents and keeps the workload of all planning agents stable. The planning agent is responsible for searching results in their local state space. Experiments show that the decentralized system can work stably and improve the composition efficiency.

The rest of the paper is organized as follows. Formal definitions for automatic service composition are given in Section 2; the composition algorithm and parallel optimization are introduced in Section 3 and Section 4, respectively; Section 5 presents the experiments and analysis; finally, Section 6 concludes the work and gives the future direction.

2 Problem Formalization

This section extends the formal definitions related to the issue of automatic Web service composition addressed in our previous work [10] in order to make them more clearly for parallel optimization.

**Definition 1 (State).** A state \( s \) is a 2-tuple \((O, P)\), where \( O \) is the set of objectives represented as 'name: type', \( P \) is the set of first-order predicates represented as 'predicate object'. Comparing to the definition in [10], the ability of expression is enhanced.

**Definition 2 (Operator).** For any service, it can be represented as an operator in the state space. An operator \( p \) is a 4-tuple \((In, Out, Pre, Effect)\), where

1. \( In \) is the set of input objectives;
2. \( Out \) is the set of output objectives;
3. \( Pre \) represents the precondition of the operator and consists of the predicates of the input objectives.
4. \( Effect \) represents the change for the global state after the operator’s execution, which consists of two categories: positive effect \( \text{Effect}^+ \) and negative effect \( \text{Effect}^- \). The positive effect consists of the predicates of the output objectives. And the negative effect consists of the predicates of the input objectives.

**Definition 3 (Coordination Operators).** Given a state \( s \), suppose that the operators \( p_1 \) and \( p_2 \) can be executed with \( s \), and the execution instances are \( p_1' \) and \( p_2' \). We say that \( p_1 \) and \( p_2 \) are coordination operators on the state \( s \) if and only if the following conditions are met:

1. Set \( o = p_1.In \cap p_2.In \), then \( p_1.\text{Effect}'(o) = \emptyset \)
   or \( p_2.\text{Effect}'(o) = \emptyset \)
2. \( p_1.\text{Effect}' \cap (p_1.Pre \cup p_2.\text{Effect}') = \emptyset \)
3. \( p_2.\text{Effect}' \cap (p_2.Pre \cup p_1.\text{Effect}') = \emptyset \)

**Definition 4 (Expansion Function).** Given a state \( s \) and a set of operators \( P = \{ p_1, p_2, ..., p_n \} \), suppose that all the operators of \( P \) can be executed with \( s \) and the execution instances are \( P' = \{ p_1', p_2', ..., p_n' \} \). If any two operators in \( P \) are coordination operators, then the following state can be achieved through the expansion function \( f_{\text{expansion}}(s, P') \):

\[
\begin{align*}
S &= s + \bigcup_{p_i \in P} p_i.In - \bigcup_{p_i \in P} p_i.Out + \bigcup_{p_i \in P} p_i.\text{Effect}'
\end{align*}
\]

Different from the expansion function in [10], the expansion function here can make a series of coordinated services executed on a certain state. Thus can increase the number of operators for each expansion and reduce the searching width, which may improve the searching efficiency.

**Definition 5 (Planning Subgraph).** Begin with a start state \( s_0 \), let a sequence of operator sets \([P] = <P_1, P_2, ..., P_n>\) function on \( s_0 \) by the expansion function in order and reach a final state \( s_n \). Then \( s_i = f_{\text{expansion}}(s_{i-1}, P_i) \) is a called planning subgraph of the sequence of operator sets from \( s_0 \) to \( s_n \).

**Definition 6 (Automatic Web Service Composition Problem).** An automatic Web service composition problem \( \psi \) is a 6-tuple \((S, S_0, S_G, P, T, \Pi)\) where

1. \( S \) is the set of all states;
2. \( s_0 \in S \), \( s_0 \) is the initial state which corresponds to the input objectives and predicates of user requests.
3. \( S_G \subseteq S \), \( S_G \) contains all possible goal states. Each state in \( S_G \) contains output objectives and predicates.
4. \( P \) is a set of operators.
5. \( T \) represents the set of reasoning relationships among the predicates.
6. \( \Pi \) represents a planning subgraph from the initial state to the goal state. It is actually a solution for the planning problem.

3 Composition Algorithm

In our previous work [10], we have proposed a composition algorithm named AWSP for automatic service composition, which supports both forward search (from input to output) and backward search (from output to input). In order to reduce the searching space and improve searching efficiency, we extend the AWSP by combing the state space approach and the planning graph approach.

The extended algorithm AWSP-E is illustrated as follows. It takes four inputs: user requested input \( s_o \), user requested output \( s_g \), a set of services \( P \) and a set of predicate reasoning \( T \); and generates a planning subgraph \( \Pi \) as the output. The main process, the heuristic function and the primary data structures in AWSP-E are the same as AWSP. We mainly make the following two extensions:

1. The searching states are extended. In AWSP-E, the...
searching states contain not only the type of the objective, but also the name and attributes of the objective. Thus can make the searching result more accurate and more possible solutions can be achieved.

(2) The operators are extended. In AWSP-E, coordination operators are applied on a certain state. Comparing to AWSP, the searching width is reduced. Then the searching space is reduced and the searching efficiency is improved.

Algorithm AWSP-E (Automatic Web Service Planner - Extension)

<table>
<thead>
<tr>
<th>Input:</th>
<th>$s_0, s_D, P, T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>$P'$</td>
</tr>
</tbody>
</table>

线

1. $Open = \phi$; $Closed = \phi$; $Graph = \phi$; $Tree = \phi$;
2. Graph$addW(s_0)$; $Tree.addNode(s_0)$;
3. calculate $f_{heuristic}(s_0)$; $Open.insert(s_0)$;
4. while $(Open != \phi)$ do
5. $c = Open.getHeadA$;
6. $Closed.insert(c)$;
7. if $(c \supseteq s_0)$ then
8. $P = path$ from $s_0$ to $c$ in $Tree$;
9. return $P$;
10. else
11. $plns = all$ operator instance which can be applied on current state $c$;
12. $ES = findAllExpansionSets(plns)$;
13. for every $P'$ in $ES$ do
14. $t = f_{expansion}(c, P')$;
15. if $(Open \supseteq t)$ then
16. Graph$addE(c, t)$;
17. $f^{new} = g_{heuristic}(c) + 1 + 10 * h_{heuristic}(t)$;
18. if $(f^{new} < f_{heuristic}(t))$ then
19. change the parent node of $t$ to be $c$;
20. label the path between $c$ and $t$ to be $P'$;
21. recalculate $f_{heuristic}(t)$;
22. replace $t$ in $Open$;
23. else if $(Closed \supseteq t)$ then
24. Graph$addE(c, t)$;
25. $f^{new} = g_{heuristic}(c) + 1 + 10 * h_{heuristic}(t)$;
26. if $(f^{new} < f_{heuristic}(t))$ then
27. change the parent node of $t$ to be $c$;
28. label the path between $c$ and $t$ to be $P'$;
29. recalculate $f_{heuristic}(t)$;
30. adjust()
31. else
32. Graph$addW(t)$; Graph$addE(c, t)$;
33. $Tree.addNode(t)$; $Tree.addParent(c, t)$;
34. label the path between $c$ and $t$ to be $P'$;
35. calculate $f_{heuristic}(t)$;
36. $Open.insert(t)$;


4 Parallel Optimization

In this section, we introduce the parallel optimization approach for automatic Web service composition based on the AWSP-E.

4.1 Parallel Composition Framework

The architecture of the parallel composition framework is illustrated in Figure 1. The parallel composition system consists of a central agent and several planning agents. The central agent is the interface between the whole composition system and the outside application system. It distributes the planning tasks to all the planning agents and keeps the workload of all planning agents stable. The planning agent is responsible for searching results in their local state space. The number of planning agents can be dynamically adjusted according to the workload of the tasks.

Figure 1 The Parallel Composition Framework

The main process of the parallel composition framework is as follows. Firstly, the central agent distributes the composition task to available planning agents when user request arrives. Then, the planning agents start to search for solutions with AWSP-E. Once a solution is found, the result is returned to the central agent and the central agent notifies the other planning agents to terminate their tasks. Finally, the central agent returns the result to the user.

4.2 Central Agent

The central agent, as the interface between the parallel composition system and the user request, needs to implement the following functions.

4.2.1 Service Registry and Management

Before the execution of the composing process, Web services from external applications should be registered in the parallel composition system. The central agent is responsible for the registry of Web services and then a catalog of services is maintained. So when a planning
agent wants to query an available service, it only needs to send the query request to the central agent, which can save much storage space for the planning agents. In general, the central agent should maintain and manage the description documents of the registered services and provide access interfaces for the planning agents.

4.2.2 Handling Composition Request
When a user request arrives, the central agent constructs an initial task and distributes it to an idle planning agent. Then the planning agent starts to plan the task. When other available planning agents are available, the central agent will extract some tasks and distribute them to these planning agents. The formal definition of a task is as follows:

**Definition 6 (Task).** A task for a planning agent distributed by the central agent is represented as a 5-tuple \( t = (Graph, Tree, Open, Closed, curState) \), where \( Graph \) is the state space graph, \( Tree \) is the generated tree of the state space, \( Open \) is the set of unvisited vertexes, \( Closed \) is the set of visited vertexes, and \( curState \) is the start state of the task.

4.2.3 Handling Composition Result
The central agent is also responsible for receiving a planning result from a planning agent and sending it to the user. If a planning agent acquires a planning result, it will send the result to the central agent. As soon as the central agent receives a planning result, it will firstly notify the other planning agents to terminate their tasks and reclaim the computation resources. Then, the central agent feedbacks the planning result to the user.

4.2.4 Management of Planning Agents
In order to fulfill the scalability of the distributed computing, agents in the parallel composition system can join or quit the system dynamically. So the central agent should be in charge of the dynamic change of the planning agents. When a new planning agent joins the system, it will notify its information (such as location) to the central agent. If a composition task is being handled, the central agent will find a planning agent with the maximum load and extract a to-be-handled task from this agent. Then this task is distributed to the new planning agent. Similarly, when a planning agent quits the system, it should notify the central agent. If this agent has unhandled tasks, these tasks will be distributed to other available planning agents or be cached in the central agent until a planning agent is available.

4.3 Planning Agents
The planning agents should response various management requests from the central agent such as load query, task callback, task termination etc. The most important responsibility for the planning agents is to handle the distributed task. The task handling is based on the AWSP-E and several tasks for multiple planning agents can be executed in parallel. The parallel algorithm for AWSP-E is as follows:

**Algorithm ParaAWSP-E (Parallel Automatic Web Service Planner-Extension)**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Graph, Tree, Open, Closed, curState</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>( \Pi )</td>
</tr>
</tbody>
</table>

1. \( localOpen = \{curState\} \);
2. **while** \( localOpen != \emptyset \) **do**
3. \( c = localOpen.getHead() \);
4. \( Closed.insert(c) \);
5. **if** \( c \subseteq s_0 \) **then**
6. \( \Pi = \) path from \( s_0 \) to \( c \) in \( Tree \);
7. **invoke** the service of the central agent to get \( \Pi \);
8. **return** \( \Pi \);
9. **else**
10. \( plus = \) all operator instance which can be applied on current state \( c \);
11. \( ES = findAllExpansionSets(plus) \);
12. **for** every \( P \) in \( ES \) **do**
13. \( t = expansion(c, \Pi) \);
14. **if** \( (Open \supseteq t) \) **then**
15. \( Graph.addE(c, t) \);
16. \( f = g + 1 + 10 * h \) (heuristic) \( t \);
17. **if** \( f < h \) **then**
18. change the parent node of \( t \) to be \( c \);
19. label the path between \( c \) and \( t \) to be \( P' \);
20. recalculate \( f \) (heuristic) \( t \);
21. **if** \( (localOpen \supseteq t) \) **then**
22. replace \( t \) in \( localOpen \);
23. **else**
24. \( localOpen.insert(t) \);
25. **else** \( (Closed \supseteq t) \) **then**
26. \( Graph.addE(c, t) \);
27. \( f = g + 1 + 10 * h \) (heuristic) \( t \);
28. **if** \( f < h \) **then**
29. change the parent node of \( t \) to be \( c \);
30. label the path between \( c \) and \( t \) to be \( P' \);
31. recalculate \( f \) (heuristic) \( t \);
32. adjust(t);
33. **else**
34. \( Graph.addNode(t) \); \( Graph.addEdge(c, t) \);
35. \( Tree.addNode(t) \); \( Tree.addEdgeParent(c, t) \);
36. label the path between \( c \) and \( t \) to be \( P' \);
37. calculate \( f \) (heuristic) \( t \);
38. \( Open.insert(t) \);
39. \( localOpen.insert(t) \);
40. **inform** the central agent that the current planning agent is available;

The algorithm takes the distributed task as the input and tries to get a planning result. If a result is achieved, the
planning will invoke the result receiving function of the central agent (Line 7). In contrast, the planning agent will inform the central agent that it is available if no result is found so that another task can be distributed to it.

4.4 Case Study

In order to make our approach understood clearly, we make up a simple but meaningful example to illustrate how the approach works. In this example, there are three categories of services related to file process as shown in Table 1. A user wants to input a doc file in English fileA and a latex file in French fileB, and output a merging pdf file of the two files which is represented in Arabic. Thus, the initial state \( s_0 = \{(\text{fileA:FILE, fileB:FILE}, (\text{doc fileA}), (\text{en fileA}), (\text{latex fileB}), (\text{fr fileB}))\} \), and the goal state \( s_g = \{(\text{FILE}), (\text{pdf f}), (\text{ar f}), (\text{merge fileA fileB f})\} \).

<table>
<thead>
<tr>
<th>Category</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>language translating</td>
<td>fr2en, en2ar, en2fr</td>
</tr>
<tr>
<td>format transforming</td>
<td>latex2doc, pdf2doc, doc2pdf</td>
</tr>
<tr>
<td>file merging</td>
<td>mergepdf, mergedoc</td>
</tr>
</tbody>
</table>

Table 1 Services Related to File Process

Assume that the parallel composition system consists of a central agent and four planning agents. Firstly, the central agent receives the request from the user, generate a initial task and distribute it to a available planning agent. The process is shown in Figure 2.

After the planning agent executes for a while, the central agent finds that there are still three available planning agents. Then the central agent extracts three tasks from the first planning agent and distribute them to another planning agents respectively (Shown in Figure 3).

The bottom side of Figure 3 shows the generated searching tree of the first planning agent. The first planning agent is exploring the left node at the moment. When the central agent send the order of task extraction to the first planning agent, the planning agent send the first three state nodes in its exploring list and the whole generated tree to the central agent. Then the central agent generates another three task according to the extracted states and generated tree from the first planning agent and distribute them to another three planning agents respectively. Now all the four planning agents are searching from different branches of the whole state space respectively.

If a available planning agent is found during again, the central agent will get the busiest planning agent and extract a task from it. Then the task would be distributed to the available agent. Thus, the whole system is kept busy and the utilization rate of the system is high.

Finally, assume that the first planning agent achieve a planning result, then the result is sent to the central agent (as shown in Figure 4). Then the central agent notifies the other planning agents to terminate their tasks and returns the result to the user.

5 Experiments and Analysis

In order to validate the feasibility of the proposed approach in this paper, we implement it in JAVA and

![Figure 2 Generating the Initial Task](image-url)

![Figure 3 Task Extraction and Re-Distribution](image-url)

![Figure 4 Final Composition Result](image-url)
conduct a serial of experiments using the datasets of WS-Challenge 2009 [11] and large-scale test sets generated by WSBen [12]. The parallel composition system we built for the experiments consists of a central agent and four planning agents. The execution environment is: Intel Core2 P7370 2.0GHZ with 4GB RAM, Windows 7, jdk1.6.0.

We firstly use the datasets of WS-Challenge 2009 to compare the performance of AWSP [10] and ParaAWSP-E. The details of the five datasets of WS-Challenge 2009 are shown in Table 2. And the comparison results are shown in Table 3 and Fig.5.

Besides the datasets of WS-Challenge 2009, we also use WSBen to generate three groups large-scale test sets. Each group has five test sets and the service number of each group is set as 100,000, 500,000, and 1,000,000 respectively. The average execution time for these test sets is shown in Table 4 and Fig.6.

From the comparison results, we can conclude that: (1) When the scale of the test set is larger, the parallel composition system can improve the composition efficiency. However, when the scale of the test set is small as Test01, the parallel composition system will take more time to get a result. The reason is that the communication between the central agent and planning agents would cost the parallel composition system some time. Therefore, for a small-scale test set, the time cost on communication is more than that on planning. So the efficiency of the parallel composition system is reduced. (2) Though the scale of the test set is large enough as Test05, the performance improvement of the parallel composition system could not reach 1/N as expected. That’s because the internal management and communication in the parallel composition system will cost some time, which can impact on the execution performance of the whole system.

From the above results, we can conclude that: (1) For the super-large-scale test sets, the parallel composition system can also improve the efficiency. (2) During the execution for these test sets, all the four planning agents are almost fully loaded all the time. Besides, with the increase of the service number, the performance improvement of the parallel composition system becomes less and less. The reason is that the number of planning agents is quite small for the super-large-scale test sets. However, for the limitation of the experiment environment, we can hardly get more planning agents. We may extend the scalability of the parallel composition system in the future work.

### Table 2 Datasets of WS-Challenge 2009

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test01</td>
<td>572</td>
</tr>
<tr>
<td>Test02</td>
<td>4,129</td>
</tr>
<tr>
<td>Test03</td>
<td>8,138</td>
</tr>
<tr>
<td>Test04</td>
<td>8,301</td>
</tr>
<tr>
<td>Test05</td>
<td>15,211</td>
</tr>
</tbody>
</table>

### Table 3 Execution time of AWSP and ParaAWSP-E

<table>
<thead>
<tr>
<th>Dataset</th>
<th>AWSP (ms)</th>
<th>ParaAWSP-E (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test01</td>
<td>47</td>
<td>102</td>
</tr>
<tr>
<td>Test02</td>
<td>188</td>
<td>197</td>
</tr>
<tr>
<td>Test03</td>
<td>312</td>
<td>243</td>
</tr>
<tr>
<td>Test04</td>
<td>579</td>
<td>398</td>
</tr>
<tr>
<td>Test05</td>
<td>891</td>
<td>689</td>
</tr>
</tbody>
</table>

### Table 4 Execution Time with Super-Large-Scale Test Sets

<table>
<thead>
<tr>
<th>Dataset Scalability</th>
<th>AWSP (ms)</th>
<th>ParaAWSP-E (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100,000</td>
<td>1,021</td>
<td>978</td>
</tr>
<tr>
<td>500,000</td>
<td>3,458</td>
<td>2,786</td>
</tr>
<tr>
<td>1,000,000</td>
<td>8,834</td>
<td>8,654</td>
</tr>
</tbody>
</table>

### Conclusion and Future Work

In this paper, we propose an approach for automatic data-intensive Web service composition, which can be executed in parallel. Firstly, we formally define the problem of automatic service composition by combing the state space search approach and the planning graph approach. Then we extend the composition algorithm introduced in...
and develop a parallel composition framework based on this algorithm. A series of experiments are conducted to validate that the parallel composition system can work stably and improve the composition efficiency.

In this paper, we do not consider non-functional factors such as QoS and Context-aware property in service composition [14]. In fact, this is more important for data-intensive web services. In the future, we intend to consider other heuristic functions to find solutions with other optimal properties (e.g., QoS) to make our approach more adoptable.

Acknowledgements

This work is supported by the National High-Tech Research and Development Plan of China under Grant 2011BAD21B02, the National Natural Science Foundation of China under Grant No. 61170033 and the project of Key Science and Technology Planning of Zhejiang Province under Grant 2012C11026-2.

References


Biographies

Shuiguang Deng received the BS and PhD in Computer Science from Zhejiang University in 2002 and 2007, respectively. At present, he is an Associate Professor in the College of Computer Science at Zhejiang University. His research interests include Service Computing, Business Process Management and Data Management. Up to now, he has published more than 30 papers in peer-refereed journals and international conference proceedings as the first author or the corresponding author. And also, he has held a number of patents for his many innovations. He is the recipient of Microsoft Fellowship Award 2005. He is member of ACM and IEEE.
Longtao Huang 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.

Bin Wu received the BS and PhD in Computer Science from Zhejiang University in 2006 and 2012, respectively. His research interests include service computing and middleware technology.

Lirong Xiong received the BS and MS in Computer Science from Central South University in 1997 and 2000. At present, she is an associate professor in the College of Computer Science at Zhejiang University of Technology. Her research interests include Service Computing, Business Process Management. Up to now, she has published more than 15 papers in peer-refereed journals and international conference proceedings as the first author or the corresponding author.