A Service Selection Workflow for Composition Using Correlation and Route Optimization

Chin-Chih Chang, and Young-Lin Lo
Department of Computer Science and Information Engineering
Chung Hua University
Hsinchu City, Taiwan
{changc, m09902026}@chu.edu.tw

Abstract—Web services have been widely accepted as a server-side technology and are the key component to cloud computing. How to effectively compose a service from a number of services is still an issue. The current solutions are either too specialized or too complex. In this paper we present a workflow models service composition as a service graph where each node is an individual service. Then the problem of service composition becomes the problem of path optimization. In our approach the services are first evaluated and ranked according to their quality of services. The highly ranked services are selected to construct a service graph determined by the correlation values among them. Finally, several service selection strategies are developed according to different correlation and path optimization policies. These mechanisms are further evaluated and validated by simulations. The experiments show the proposed methods are feasible and not complex.

Keywords—Correlation; Web services; service selection; path optimization; service composition.

I. INTRODUCTION

Web services have become a popular server-side technology since their inception in 2001. They play the core role in cloud computing as services are designated in three forms: infrastructure, platform, and software. They have a promise that service composition can be automatic. As more services are available in clouds, how to effectively compose a service from a number of services to meet a variety of demands has become an important issue. Until now, though quite a number of approaches have been proposed, an effective and less complex solution is not available.

Web services contain three basic standards: Simple Object Access Protocol (SOAP) for service communication, Web Services Description Language (WSDL) for service description, and Universal Description, Discovery, and Integration Infrastructure (UDDI) for registry and discovery. Business Process Execution Language (BPEL) is designed for service composition. In the Web services architecture service requestors request services provided by service providers through a service broker or registry. To response a suitable request it usually goes through Web services discovery, selection, and composition. Service composition is so far the most difficult task to achieve. Service composition can be either static or automatic. In static composition services are selected in the design phase. In dynamic composition services are selected in the run time. How to effectively compose a service from a number of services is a still an issue.

Correlation-aware and graph-based selection have been proposed to support service composition [1, 2, 3, 4]. It is pointed out highly correlated services can be integrated into the composition process more seamlessly [1, 2, 3]. Web service composition can be modeled as a direct acyclic graph [2, 4]. Most of these approaches are commonly theoretic and a practical solution is still an open issue. In this paper we combine correlation and graph-based approaches to present a feasible service selection mechanism for composition.

This paper is organized as follows. Section 2 discusses the related work. Section 3 describes the system architecture and workflow. Section 4 illustrates the system implementation and experiments. Section 5 presents the conclusion and future work.

II. RELATED WORK

A. Correlation-aware Service Selection

When a service request is sent to a service registry, the request will match against a list of services and the registry will return the services that fulfill the request through a series of operations: service discovery, selection, and composition. Service discovery is considered as the fundamental one and its main task is finding the services which meet the functional criteria of a user’s request [5, 6]. The discovery process is mostly keyword-based search which returns a list of candidate services. The discovery result is later refined by the selection process. The common task of selection process is to identify those services which meet nonfunctional properties, also known as Quality of Service (QoS) [5, 6]. A recent survey on service discovery and selection can be found at [5] and [6] respectively.

Quite a number of QoS-based selection methods have been proposed. Most of them rely only on QoS and are not able to handle the correlations among services. But in fact services are not independent among each other and more researchers have pointed out the correlations among services are important especially for service composition [1, 2, 3]. In different approaches different correlations are defined such as flow, compatibility, QoS, semantic, statistical, contextual, transactional, and data driven correlations [2, 3]. In this paper, we focus on the QoS correlation indicating dependency between a service and a set of services and can affect the quality of service composition [1, 2]. QoS correlation is decided by whether a service can enhance or decrease QoS of another service. For example, a stock trading service may charge less fee if trading is done online (positive correlation),
but more fee if trading is helped by a stockbroker (negative correlation) [2].

B. Service Composition and Model

One of highly prospective benefits of Web services is service composition which facilitates the software reuse. Service composition is required when a request is too complex to be accomplished by a single service. The common strategy is to decompose a complex request into a sequence of simpler requests each of which can be fulfilled by a single service [7, 8, 9]. Service composition is classified into static and dynamic compositions. In static composition the whole composition process is known before composition. The main issue is to arrange the appropriate services based on the requestor's criteria. Most recent researches focus on dynamic service composition in which the composition process is determined in the run time. How to dynamic decompose a request and then select suitable services are still a challenge.

Service composition process can be specified in three phases: composite service definition, service deployment, and service execution [9]. A composition scheme must satisfy four requirements: connectivity, nonfunctional QoS properties, correctness, and scalability [7]. Workflow composition and AI (Artificial Intelligence) planning are two common approaches for service composition [10, 11]. OWL-S (Ontology Web Language – Semantic) is a language for service composition. In the workflow approach service composition is modeled as a workflow of processes. In the AI planning approach an AI planner orchestrates service composition [11].

Service operations can be modeled as graphs, finite state machines (FSMs), and Petri-nets for analysis. The graph model is most common one in which each node is a service. In static composition the graph is constructed before composition. In dynamic composition the graph is constructed towards the goal of composition [4, 13]. The execution plan then transforms to a problem of planning a route to the destination. The current graph-based approaches are either too theoretic or too complex from our point of view. Effective dynamic Web service composition is still a highly complex and challenging task. In this paper we propose a selection framework for composition which is feasible and not complex. A recent survey of reliable dynamic service compositions can be found at [12, 13].

C. Route Planning and Optimization

Once the service operations can be modeled as a graph. We can apply the method for the graph analysis to service operations. Route planning and optimization in graphs are classic mathematical problems which have evolved from notable Seven Bridges of Königsberg to Dijkstra’s algorithm. The similar approaches can be applied to service operations. Route planning and optimization aim at computing the most cost-effective route through designated nodes by minimizing either distance or time. In terms of service operations the distance becomes the weighting between services which can be a measure of correlation, input-output compatibility, composability, etc.

Since route planning and optimization are classic problems, various solutions are available. When the problem size is small, optimal solution can be archived. But when the problem size is large or there is no optimal solution, usually near-optimal solutions are acceptable. The common practice is trying the optimal solution first and then the near-optimal solution when calculation is running out of computing resource or time constraint. Linear programming, dynamic programming, branch-and-bound, and branch-and-cut are adopted for optimal solution (exact approach). Heuristic methods are used for near-optimal solution (approximate approaches). Heuristic methods fall into three categories: construction, improvement, and hybrid. Each category can be further divided and is quite diverse. The interested reader can refer to [14].

The route planning method presented in this paper is a variation of the approximate construction-based approach in which the route is constructed level by level until the goal is reached. For route optimization the Prim method [15] is adopted. For service selection and composition, some revision is required to adapt the above approaches because of the inherit properties of services. In the next section we will explain it in details.

III. SYSTEM ARCHITECTURE AND WORKFLOW

A. System Architecture and Workflow

The proposed system is composed of five modules where query and provider interfaces reside on the client side and the rest them on the server side as shown in Figure 1.

- Query Interface: A Web-based interface for a user to query and invoke the service.
- Provider Interface: A Web-based interface that facilitates service providers to register their services and store the correlation value from 1 to 10 specified by a professional.
- UDDI repository: This module contains a UDDI registry and a database that stores services.
- Filtering module: This module enhances UDDI with the discovery and selection mechanism including keyword matching, filtering, and qualifying.
- Route Optimizer: This module further enhance the selection mechanism by correlations including service classification, graph construction, and route optimization.

In the proposed architecture, the request will go through three phases. In the first phase a list of services are retrieved from the service database through UDDI and those which do meet the functional criteria are filtered out. In the second phase the services are filtered by the keyword (semantic analysis) and ranked and qualified by the quality index (Qi) (QoS) as defined in Filtering Large Qualifying Rest (FLQR) model [16] and weighting entered by the user as shown in Table 1. In the third phase we enhance the method presented in [17] for route planning and optimization by the Prim’s algorithm [15]. The final results will send back to the requestor through the query interface.
TABLE I. LIST OF TERMS FOR QUALITY INDEX

<table>
<thead>
<tr>
<th>Terms</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability (Qav)</td>
<td>Qav = Nsuccess/Ntotal</td>
</tr>
<tr>
<td>Reliability (Qre)</td>
<td>Qre = Ncomplete/Nsuccess</td>
</tr>
<tr>
<td>Performance (Qt)</td>
<td>Qt = 1 – Nt (Taverage)</td>
</tr>
<tr>
<td>Quality Index (Qi)</td>
<td>Qi = Wi Qav + Wj Qre + Wk Qt</td>
</tr>
<tr>
<td></td>
<td>Wi + Wj + Wk =1</td>
</tr>
</tbody>
</table>

- Nsuccess: Times of successful searches
- Ntotal: Times of total searches
- Ncomplete: Times of completion
- Taverage: Average time of execution

B. Graph Construction

The selected services in the second phase are first classified as shown in the example of Figure 2. Nine services are classified into three categories which are specified by the service providers. Each category occupies one column where each layer has one service. The services with the largest number of services are put in the middle column, the second ones in the left column, and the third ones in the right columns. Then a complete service graph is built by connecting each node with an edge.

But a complete graph is not efficient for route planning and optimization because some redundant edges are either infeasible or useless for service selection. Evidently, it is necessary to reduce the complexity of the graph by decreasing the edges. In [17] the authors proposed a number of strategies to decrease the number of edges and adopted greedy and correlation-aware methods for path finding. These methods reduce the complexity of the graph and make route planning and optimization faster and more effective [17].

Here, we introduce the correlation graph construction strategy because it shows better performance than other methods in [17]. Whether an edge between services is constructed is determined by the correlation values between them. We have found the larger the value is the less edges the graph has and when the base value is 5 the graph has proper edges to connect nodes. As shown in Figure 3 WS1(8, 9) is a meal type of services and has a correlation value 8 regarding to the accommodation type of services and 9 regarding to the activity type of services respectively. An edge is built when a service of different categories at the different layers both have a correlation value larger than the base correlation value. Based on this rule there is no edge between WS1 and WS2 because the correlation value 4 at WS2 is less than 5. Hence, two edges are built in Figure 3 and the service graph is constructed from Figure 2 as shown in Figure 3.
C. Route Optimization

Once a service graph is constructed, the last phase of the service selection workflow becomes finding a route/path from the starting point to the end. To facilitate route planning and optimization the distance between nodes/services need to be defined. In [17] the distance between two services residing at two nearby vertical layers or horizontal categories is assigned to 1. Sure, this distance is adjustable. For example, if the distance between layers has more effect on the service selection, then we can assign higher distance between layers. In [17] the shortest path and correlation-aware path finding are proposed. In the shortest path each time a shortest path is chosen until reaching the end. This is a local optimal solution. In correlation-aware path finding the correlation distance ratio (α) is introduced as shown in the equation (1):

\[
\alpha = \frac{\text{correlation between the targeted services}}{\text{distance between two services}}
\]  

\[\text{(1)}\]

Each time the path with the highest correlation distance ratio is chosen until reaching the end. The arrow lines shown in Figure 4 indicates the path by correlation-aware path finding. In this paper we enhance the previous work for two purposes: enhancing the quality of selected services and reducing searching time. For the first purpose, we design quality factor (QF). In the process of finding the optimal route we consider not only correlations but also quality index and define the QF as shown in Equation (2).

\[
\text{QF} = \alpha + Q_i \times \text{Weighting}
\]  

\[\text{(2)}\]

For the second purpose, our approach is to reduce the size of the graph without loss of the high quality of services. We adopt Prim’s algorithm which is a greedy algorithm efficient for finding a minimum spanning tree for a connected weighted undirected graph in which a minimum spanning tree containing all nodes with total minimum weight [15]. Considering the graph built by correlation graph construction with the hierarchical constraint, the service graph and the optimal route in the arrow lines are illustrated in Figure 5 after applying Prim’s algorithm and quality factor.

![Service graph after optimization by Prim’s algorithm](image)

Figure 5. Service graph after optimization by Prim’s algorithm

IV. EXPERIMENTS AND RESULT ANALYSIS

In order to validate the proposed method we develop a simulation environment to compare it with the previous work.

A. Simulation Environment

The simulation environment is developed with hardware and software listed in Table 2. In our simulation 12000 travel services of three categories are randomly generated including 3970 accommodation, 4104 meal, and 3926 activity services in which each service is randomly assigned a correlation value.

<table>
<thead>
<tr>
<th>TABLE II. DEVELOPMENT ENVIRONMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
</tr>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>Motherboard</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td><strong>Software</strong></td>
</tr>
<tr>
<td>Operating Syste</td>
</tr>
<tr>
<td>Development</td>
</tr>
<tr>
<td>Programming Languages</td>
</tr>
</tbody>
</table>

In the above 12000 services 300 services are designated as high-quality services which have the high correlations with other services and are consider to be more suitable for composition. The number of these high-quality service can be changed when correlation values is set at a different value. We have run simulations at 75, 150, 300, 600, 1200, 7200, and 1200 services in which the numbers of high-quality services are shown in Table 3.

<table>
<thead>
<tr>
<th>TABLE III. NUMBER OF SAMPLE SERVICES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of services</strong></td>
</tr>
<tr>
<td>75</td>
</tr>
<tr>
<td>150</td>
</tr>
<tr>
<td>300</td>
</tr>
<tr>
<td>600</td>
</tr>
<tr>
<td>1200</td>
</tr>
<tr>
<td>7200</td>
</tr>
<tr>
<td>12000</td>
</tr>
</tbody>
</table>

B. Result Analysis

The precision and recall rate are used to evaluate the proposed method. The precision rate indicates the fraction of target document in all retrieved documents while the recall rate indicates the fraction of the target documents that are successfully retrieved as shown in Equation (3) and (4) [18].

\[
\text{Precision Rate} = \frac{\text{number of target documents selected}}{\text{all the selected documents}}
\]  

\[\text{(3)}\]

\[
\text{Recall Rate} = \frac{\text{number of target documents selected}}{\text{all the target documents}}
\]  

\[\text{(4)}\]

The experimental results are shown in Figure 6, 7, and 8. Each value is calculated by averaging values of running 100 times. The summary of the comparisons is shown in Table 4. It shows that correlation-aware path finding with Prim’s optimization uses less time and has better precision without loss of recall though precision and recall are inverse most of time [19].
V. CONCLUSION AND FUTURE WORK

In this paper we present a Web service selection workflow for composition based on correlation and route optimization. Through search, filtering, and optimization phases a set of high-quality services are recommended. The method is validated by experiments. The experimental results show the method performs better than the previous work in terms of time and selected services.

Currently, more optimization methods are investigated and more filtering strategies are studied. Next, we plan to collect user’s feedbacks and further improve the selection workflow by them. We believe user’s feedback will allow us to find more services that users really prefer.

ACKNOWLEDGMENT

This work is partly supported by the Ministry of Technology of Taiwan under the grant number NSC 101-2815-C-216-011-E and NSC100-2218-E-216-001.

REFERENCES


