A Quality of Service (QoS) Driven Automatic Service Composition Algorithm

Chun-guang ZHANG¹, Qiao SUN¹, Lei SUN¹, Si-ning WANG¹, Xiao TONG¹, Yi-peng Su², Zhi-dan LI¹ and Wan-tao LIU²,*

¹Beijing GuoDianTong Network Technology Co., Ltd., Beijing, 100070, China
²Institute of Information Engineering, Chinese Academy of Sciences, Beijing, 100093, China
³Luoyang Institute of Information Technology Industries, Luoyang, 471000, China

*Corresponding author

Keywords: Cloud, Automatic service composition, Dynamic QoS.

Abstract. In complex and large scale systems, such as service cloud, huge number of service resources makes lots of user to raise personalized complex service processing request. At such sitation, we need high performance and real time service composition system to meet functionalities requirement, and at the same that to maximize the resource utilization of system. We have two major challenges to build such high performance composition system: the ability to parallel processing large scale service request in the front end; and the ability to monitoring and fetching dynamic QoS to support QoS driven service composition planning calculation in the back end. We propose a three-layers filtering based QoS driven automatic service composition algorithm. The experiments show that our algorithm significantly improves the service composition performance.

Introduction

The rapid development of the current Service-Oriented Computing (SOC) and Service-Oriented Architecture (SOA) enables programs of different organizations and individuals to be used by people anywhere. In this context, a single atomic service within a single application and organizational boundary often fails to meet the complex needs of modern enterprise processes and the ever-changing needs of users. Therefore there is a need to combine multiple interoperable atomic services to accomplish more intensive functions. With the development of cloud computing technology and large-scale data center, the dynamic QoS-driven automatic service composition algorithm which can dynamically bind computing and storage resources to the specific implementation of the workflow to achieve the flexible sharing and efficient utilization of computing resources has become a hot research topic in recent years. Cloud computing brings in new opportunities to service computing technology. First of all, the service definition has been expanded in width. Traditional legacy applications and various hardware and software resources can be encapsulated into services. Such services can be IT, software and Internet related or can be any other services. Secondly, with the development of some virtualization technologies, the service definition has also been further explored in depth, such as infrastructure as a service (IaaS), platform as a service (PaaS), software as a service (SaaS), and process as a service (PaaS). Therefore, the cloud computing environment will become a huge container for large-scale services. The issues to be addressed by the automatic service technology will also shift from interoperability and stability to scalability and high efficiency.

In response to the above issues and requirements, although Web services technology provides an integrated and interactive mechanism for heterogeneous, autonomous and loosely coupled distributed applications, a single Web service function is often too simple and limited to meet some practical applications. As a result, there is a need for composition of existing single Web services. On this basis, in order to meet the users’ requirements on system performance, the Quality of Service (QoS)-driven automatic service composition technology has also been proposed to produce...
more complex, more powerful Web services to support a variety of application needs.

**Related Works**

For the services with the same functions, different providers may provide different quality of service, such as different service prices and service response time. Therefore, how to select a service provider properly and make an optimal solution for the overall operation plan to meet users’ QoS-constraint objectives have been the most important research subject on Web service composition. Unfortunately, in service composition, the QoS-based service selection issue is similar to multi-resource, multi-attribute and multi-choice issues problem, which is modeled as Multidimension Multichoice Knapsack Problem (MMKP), which is NP-hard. There are many different solutions to this problem.

The earliest work on formal modeling of service selection is literature [1]. The work uses status diagrams and DAGs to model composition issues. Status diagrams are used for an intuitive demonstration of service composition, DAG is used for modeling execution paths. This approach is able to aggregates similar services to form a WS-Community. During the composed services running process, if conditional branches are not included, there is only one execution path, otherwise there are multiple ones. After instantiating a task on the execution path, namely, selecting an appropriate service, it will become an execution plan. Obviously, if we only choose optimal option for one attribute, we will choose the best service on each node on the execution path so that we will find out the globally optimal solution. Therefore, the main issue is about multiple attributes.

The paper finally proposes three methods of service selection. The first one considers only a single execution path and normalizes weighted sum of multiple attributes, and then gives the optimal solution after scoring each execution plan. Secondly, in the case of multiple execution paths we propose a greedy strategy, that is, if a task belongs to only one execution path, then the task will choose the selected services on this execution path. If a task belongs to multiple execution paths, then we will choose the most frequently selected WS. However, the article does not give the proof that the globally optimal solution can be found out. Thirdly, we can use integer programming namely the IP method to find out the optimal solution. We can input IP with variables, objective function and constraints. For properties, such as reliability and availability, you first have to linearize. The work mentioned above is based on the SELF-SERV [2,3] platform to achieve the algorithm discussed in this thesis. To use IBM Web Services Toolkit 2.4 (WSTK2.4) as a tool for web services development to call IBM's Optimization Solutions and Library (OSL) for IP issues. SELF-SERV is a rapidly developed and implemented service composition platform developed by the University of New South Wales and the University of Queensland. It uses State Chart to represent the service composition model, whose state transition conditions are indicated using the ECA (Event-Condition-Action) rule. SELF-SERV uses P2P service composition execution mode, which generates its coordinator for each activity and each coordinator can maintain a routing table of the service composition that records the predecessor and successor positions of the activity. However, SELF-SERV only supports keyword-based service query rather than context.

Multi-Choice Knapsack Problem (MCKP) can be modeled as the service composition method and three algorithms including exhaustive search, dynamic programming and pisinger. However, this approach is weak in modeling transmission delays and costs. Because different service instances belonging to the same task can be provided by different providers and different providers can be in different networks, there is a difference in transmission delay and cost between service instances, but in the composition approach, we can only assume that the transmission delay and cost between two service instances is the same.

We can use DAG in the graph to model and map some parameters such as cost and benefit to edges by conversion. We can model issues as the shortest constrained ones in graph theory, using Constrained Bellman-Ford (CBF) and Constrained Shortest Path (CSP) algorithms. The CBF algorithm is exponential.

[4] Research on automatic service selection is mainly based on the LLAMA platform. It is an SOA middleware framework that aides the lifecycle of large SOA-based IT systems management.
processes, including automated service composition, runtime monitoring, problem diagnosis and continuous process re-configuration. Among them, the LLAMA ESB is the core of the whole system. Based on the open source project called ESB-Mule, LLAMA ESB implements the function of dynamic routing by extending the Mule ESB and adding routing and message interception to the system. The QBroker component has been developed by the QCWS module to implement different service composition algorithms, including single-QoS attributes and multiple QoS attributes and different composition modes. Reliability management module is mainly responsible for service-based runtime monitoring research and service process error diagnosis based on Bayesian networks. The trust and reputation management module is responsible for evaluating service reputation and combining service reputation with context to enhance service reliability.

There is a lot of work that takes the planning idea in artificial intelligence into composition service technology to achieve an automated service composition. Generally speaking, planning can be described as a set of possible states, a set of executable actions and a set of state transition rules. The goal of the plan is to find a group of action sequence from the initial state to the target state. In AI-based service composition, the initial state and the target state are defined by the requirements of combined services. The action is a set of available component services. The state transition rule defines the prerequisite and effect of the service function of each component. Therefore, the process of such service composition is to find a set of services from the optional component services so that the function of the composite service can meet the requirements of the composite service. [5] To propose an automatic modeling algorithm of composite target decomposition, composite reasoning and composite service model by using Planning Domain Definition Language (PDDL) on the basis of rule-based programming and combining with the context of the semantic Web computing. [6] Calculus is a first-order, logical language used to represent a dynamically changing world. It is a formal predicate calculus about states, actions, and actions for state outcomes. [7] To use Golog as a language for automatically building a web services composition, solving Web services composition problems by providing higher-level procedures and user-defined constraint descriptions. Golog is a formal method representing inference issues. The main idea of this method is to use software agents to reason about services so that Web services discovery, execution, composition and interaction can be performed automatically. Users' requirements and constraints can be described by the first-order predicate language of scenario calculus, so that inference in Web services space can be used to discover a set of Web service sequences to satisfy service requests.

Reference [8] provides a method of mapping specific service composition execution path selection problems to graph path selection. In the process of transforming service selection into graph, the method considers the concrete service instance as the vertex of the graph, the execution relation between the instances as an arc, and the comprehensive utility obtained by executing the service instance becomes the weight of the corresponding arc. At the same time, the original structural structure of the composite scheme is retained in the map, that is, the preservation order, parallelism and other control structures. In this way, the problem of optimal selection of service composition paths becomes the problem of solving the path from the source node to the target node in a directed weighted graph. Path optimization selection problem. This article provides a series of algorithms to solve the synthetic execution path, and provides a comprehensive analysis and comparison of the selection effects of various algorithms.

Literature [9] proposed the optimization selection method of mapping synthetic flow application problem to graph path in P2P operation structure and proposed two key algorithms: global center algorithm and distributed local algorithm. This method compares the proposed algorithm with the traditional graph path selection algorithm such as Dijkstra-composites algorithm in detail, which shows that the proposed method has been improved in terms of reducing the complexity and success rate.

**Problem Definition and Solution**

Regarding the above analysis, we first formally define the problem to be solved, and then give our solution.
The definition of auto service composition problem is to give a service set \( G \). If there is a composition request \( R \), it has initial set of input parameters \( R_I \) and set of required output parameters \( R_O \). The service composition issue can be defined as to find a service subset \( \text{Set}(w) \) in the service set \( G \). Any one of the services \( W \) belonging to \( \text{Set}(w) \) can be triggered in the runtime environment initiated by the request \( R \), and \( R_O \cap \text{Set}(w) = R_O \). The intrinsic connections between these services are Sequence, Combine, and Split structures as shown in Figure 1. These services provide the parameters of each other calls, and these organizations combined into a service process can be called.

In general, the quality of service attributes \( Q \) to be considered include response time, throughput, accountability and availability. Atomic services in the service-driven automated service composition not only need to satisfy the triggering and interoperability constraints between services in the automatic service composition. Their combination also determines the overall quality of service.

Service Quality-Driven Auto Service Composition Problem Definition: Given a service set \( G \), any one of the services \( W \) is a triplet \( \{W_I, W_O, W_Q\} \), where \( W_Q \) is the service quality attribute of the \( W_Q \) service. Find a service subset \( \text{Set}(w) \) in the service set \( G \), any service \( W \) belongs to \( \text{Set}(w) \) is triggered in the operating environment initiated by the request \( R \), \( R_O \cap \text{Set}(w) = R_O \), and its combination Results Composite Service (CS) is the best quality of service.

Service Filtering Algorithm Based on Graph Accessibility

The reachability of graphs has been extensively studied in the field of graph theory with the aim of solving the problem of how to use the least time complexity in a given graph \( G = \{V, E\} \) to answer if a node \( U \) in a graph can be connected Go to another node in the diagram.

Common methods are Chain Decomposition Approach and Tree Cover Approach. Both of these algorithms need to find all the strongly connected components in the original image and then aggregate these connected subgraphs into a single point of the abstract graph, and then establish the interconnected index on the graph node. Based on these two methods, Agrawal and Wang propose Dual-Labeling Approach to solve the problem that the time complexity of tree coverage algorithm is \( O(n) \) and the time complexity of indexing is \( O(n^2) \). The efficiency of sparse graphs in query time and index space can reduce the complexity of index construction to \( O(n + t^2) \) and reach the query time of constant. Other methods include Label + SSPI, GRIPP, 2-HOP Labbeling and others.

Our filtering algorithm is based on the Path Tree Cover algorithm proposed by Ning Tuan. Their approach reduces the time complexity of building an index to \( O(m + n \log n) \) and guarantees \( O(1) \) reachability query time complexity. So help to reduce the system initialization time and filtering time overhead.
Reachable filtering, which is based on reachability of graphs, works well for sparse interoperability graphs. The real idea is that if a service does not have a path to the output parameter of any combination request, then that service does not contribute to the combined computation. However, when the connectivity between very dense services of interoperability graphs is very strong, the filtering effect of graph-based reachability filtering algorithms is not ideal, and the algorithm cannot guarantee that the filtered services must be able to be triggered. Therefore, we finally chose Counting-based algorithm breadth-first traversal of the entire interoperability map, find a certain set of services can be triggered.

Service Quality Driven Automatic Service Composition Algorithm Based on Dynamic Programming

Dynamic programming is a method used in mathematics and computer science to solve optimization problems involving overlapping sub-problems. The basic idea is to decompose the original problem into similar sub-problems, and then find the solution to the original problem through the solution to sub-problems in the process. The idea of dynamic programming is based on a variety of algorithms. Well-known examples of application algorithms include shortest path problem, knapsack problem and network flow optimization problem.

Dynamic programming is most effective in finding the optimal solution to the condition where there are many overlapping sub-problems. In order to avoid solving these sub-problems many times, their results are gradually calculated and saved so that problems can be solved from simple to global. This time dynamic programming saves recursive results and therefore does not take time to solve the same problem next time. Moreover, the dynamic programming can only be applied to the problem with the best sub-structure. The optimal sub-structure means that the partially optimal solution resolves the global optimal solution.

The solution to the optimal solution to service composition computation is to find a set of parameters that a DAG can obtain the portfolio request with the minimal cost. Therefore, the optimal solution of each parameter is a sub-problem of the whole composition issue and has the best sub-structure property. Moreover, in a large-scale service collection, many services often require the same type of parameters. Therefore, a computing structure with optimal parameters can be used by multiple services, as a result it has the property of overlapping sub-issue.

As shown in Figure 2, we use a hash table to store the available service parameters during the calculation. Since multiple services may produce the same service parameters at different costs, there are a number of updates to the operation of the hash table, in addition to adding operations, including updating its most value and generating service pointers for this parameter used to find all the services backtracking after the composite calculation is completed and to build the DAG as a combined result.
Because we have performed service filter calculations before making a composite, we know the dependencies between the service set and the service being triggered. Therefore, we can topologically order triggerable services so that the optimal value of delayed services (minimum response time, maximum throughput) to the parameters required by the service is the optimal response time and throughput, thus preventing Repeated data update operation.

The calculation process is as follows: First, put all the input parameters of the combination request into the hash table, and then remove the service from the service set that already has the topological sequence number, and search the hash table for the best corresponding parameter according to the input parameter it needs Value, calculate the service's own optimal value and then update the state of its input parameters. If there is no parameter in the parameter hash table, it is added directly. If there is, you need to check whether the optimal value of the parameter needs to be updated. The combined calculation process is looped in this way until all service processing ends.

To support multithreaded concurrency, we use the ConceptWrapper, a guarantee class, to record information that needs to be independent of a composite request, such as optimal response time, optimal throughput, and service pointers that produce these two optimal values. And the parameter hash table is also independently owned by each thread. Through this structure, in the design of our concurrent system, we only need to let different processing threads pay attention to the current combination of calculated variables, and do not need to do other extra operations during thread switching. In order to reach thread safety, we need to pre-allocate enough memory for each thread at system initialization. If you create a thread for each request for processing, it will inevitably lead to the exhaustion of system resources. This is why we did not use multithreaded / procedural patterns later in system design.

**Time Complexity Analysis**

The complexity of the entire composition calculation process consists of three parts, the combination of filtering, topological sorting and dynamic programming. The time complexity of the filtering algorithm is $O(n)$. Assuming that the number of services in the filtered interoperation graph is $sn$ and the number of edges is $ln$, the complexity of topological ordering and the average time complexity of dynamic programming are $O(ln)$ and $O(sn)$. Since the filtered $sn$ is far less than $n$, the average time complexity of the combined computation is $O(n + ln)$.

In the worst case, the services in the service set can all be triggered (ie, $sn$ equals $n$) and there are $m$ edges in the whole graph. The time complexity of topology ordering is $O(n)$. Suppose there is a worst combination result, the result of the combination is $n / k_1$, and the number of times each layer needs to update the parameter set is $m / k_2$, the total time complexity is $O(m * n / (k_1 * k_2))$. After
the constants $k_1$, $k_2$ are omitted, then the worst-case time complexity of the algorithm is $O(m \times n)$, which is the algorithm's time on-line.

The spatial complexity of the algorithm is the sum of the number of filtered services and the number of parameters. In the worst case, all services can be triggered and all the parameters can be generated. Assuming there are $n$ services and $K$ parameter ontologies in the portfolio, the worst-case complexity of the algorithm is $O(n + k)$.

**Experimental Comparison**

In this section, we validate the performance of our algorithm mainly through WS-Challenge’s Benchmark, including the scalability in terms of increasing the number of parameters like service count and process complexity.

The WS-Challenge is an international competition sponsored by the Natural Science Foundation of the United States. Organizers arranged the competition to solve the core problem in service computing - automatic service discovery and composition.

WS-Challenge provided a service quality-driven automated service composition algorithm test platform. On WS-Challenge Benchmark, we can configure a number of attributes including the number of services and the number of parameters ontology to generate different types of test sets. Finally, a verification program verifies the functional correctness of the composition results and the quality of service of the process. All of the following experiments are run in computing environment with 2.53GHz-dual-core and 2G-memory at Windows XP.

We named the algorithm QSynth and the comparison algorithm we used was based on the same platform Peter and Yan's solution. From the left figure in Figure 3, we characterize the performance of QSynth, Yan, and Peter as the number of services increases. Obviously, our algorithm still has the fastest processing power and we have less than 200ms processing time in these groups of test sets. From the figure, we can see that Peter can not guarantee the optimality of the combined result.

![Figure 3. Experimental results of increasing number of parameters ontology.](image-url)
From the left graph in Figure 4, we demonstrated the performance of our method together with Yan and Peter’s as the number of parameter ontologies increases. It can be seen in the left figure that the composition computation time does not increase with the number of parameter ontologies because in a test set, if the parameter types (the number of ontologies) are small, it means the interoperability between services Link density is higher. Therefore, in the condition of the same service number and call depth, the fewer the number of ontologies and the more services are triggered, the higher the cost of natural calculation is. This also shows in the left figure, when the number of parameters ontology is 5000, why the three composition algorithm’s query time is the highest. Referring to the left and right two figures, we can see that the performance of our composition algorithm is the highest and the optimal solution can be guaranteed. Yan’s algorithm could guarantee the optimality of the result, but the computation time was longer than that of Peter’s while Peter’s result could not be optimal.

Conclusion
This thesis introduces the QoS-based automatic service composition algorithm by self design, and introduces the corresponding algorithms from three aspects including the pre-combination filtering method, the sub-optimization in dynamic programming and the optimal combination result respectively. The time complexity of the algorithm is given based on the above. In the filtering algorithm, we introduce the service filtering algorithm based on the reachability of graphs as well as the filtering algorithm based on counting and give the scenarios that they apply respectively. After analyzing the issue property, we propose a solution based on dynamic programming to improve the speed of computation by preprocessing techniques of topological ordering. In the stage of generating composition results, we put forward the concept of critical path. Based on the information of critical path, we propose a greedy process compression algorithm.

Finally, we compared the performance of our algorithm with that of Peter and Yan in the WS-Challenge Benchmark environment. The experimental results show that our algorithm can not only find the optimal solution, but keep the record of offering the composition result within the shortest calculation period among the three algorithms. Therefore, the algorithm we propose is the most efficient in composition.

Acknowledgments
This research was financially supported by Science and Technology Project of the State Grid Corporation of China (SGGSXT00YJJS1600102) and the State Grid Information & Telecommunication Group CO., Ltd. (SGITG0000KXJS2016040).
References


