Abstract

The SOA model brings new benefits to software design and architecture by enabling re-use and sharing of services. Due to the uncertain of web service, it is important to make sure that the selected service is always reliable and available. In this work, we propose an allocation scheme that minimizes the response time and cost of the solution subject to reliability and availability constraints in terms of expected value. The algorithm proposed in this paper aims to discover services with high QoS performance, and reduce the execute time at the same time. Finally, we proves the advantage of the new algorithm by comparing the time obtained by our proposed algorithm with the one achieved by other algorithm.

1. Introduction

With the development of the distributed processing technology, the existing theory in the face of larger and more flexible application show significant shortcomings. Web service solves the heterogeneous distributed computing, as well as code and data reuse, which has a high degree of interoperability, cross-platform and loose coupling characteristics. For the Internet, software and integration between the re-use provides the technical basis which can reduce the duplication in the software industry, and increase utilization of the software [1]. Therefore, more and more academic as well as industrial communities pay attention to the research of Web services technology [2].

In general, the function of a single service is limited, it is difficult to satisfy practical needs. Therefore, it is necessary to compose individual service to complete complex task. In the service composition process, it is important to ensure service quality and reliability. Thus, how to select and compose services which can meet quality of service (QoS) requirements and get the minimum cost is meaningful investigation. To solve the optimal service selection problem, we must first provide a business process, and take into account the interaction properties of service composition model. Then, we must determine the service selection that satisfies the QoS and cost constraints [3].

This paper is supported by Natural Science Foundation of China (No. 70940008) and Specialized Research Fund for the Doctoral Program of Higher Education (No.200801510001).
Recently, a lot of helpful research work on service selection and composition has been done by various researchers. For instance, Tsesmetzis et al. [4] studied the problem of service providers that receive multiple concurrent requests for services demonstrating different QoS properties. The paper focuses on two of these properties and addresses the case where the available services (or service versions) require specific bandwidth and are offered at a certain cost. Wang et al. [5] proposed a fuzzy-based UDDI with QoS support. The proposed method tries to consider not only the objective factors described by service providers but also the subjective information with trustability evaluations from users who use those services. Genetic algorithm (GA) is adapted to learn user preferences, and fuzzy logic is applied for making decisions. E. Gerede et al.[6] used a framework to integrate activity processing costs into the delegation computation and to have services with bounded storage as opposed to finite storage. And investigate the problem of efficient processing of service requests in service communities and develop polynomial time delegation techniques guaranteeing optimality. However, in spite of the important theoretical advances that have been achieved in these areas, it is still difficult to find an efficient way to compare Web service based on QoS. Current proposals use exact algorithms or heuristics to solve the QoS-aware (optimal) service selection problem for each request, whose exact solution has an exponential complexity. In this paper, we propose an algorithm to find the optimal solution which can better meet the service requester’s requirement, and have less cost at the same time.

2. Service Analysis

2.1 The analysis of service attributes

The quality of Web service (QoS) is non-functional properties (including service cost, security, reliability, usability, runtime etc.) to describe the degree of satisfaction of users to Web service. Four general service attributes are chosen in this paper, which are response time, reliability, availability, price [7].

Response time can be defined as time taken to display response on sending a request. It comprises the execution time and waiting time which can be flawed due to uncertain network fluctuations.

Reliability is the ability of a service to perform its required functions under stated conditions for a specified period of time. This ability which follows Poisson process can be quantified by the probability of success in a service implementation.

Availability means the ratio of the total time a service is capable of being used during a given interval to the length of the interval. A simple representation for availability is as a ratio of the expected value of the uptime of a service to the aggregate of the expected values of up and down time.

Price is the cost of service for a request. The service with higher cost provides more complex functionality.

The Optimal Service Selection is to find a service selection with less price and response time, but higher reliability and availability, as illustrated below.

\[
F(a) = \min(T(a) + P(a) + R(a) + A(a))
\]  

Subject to:

\[
E(R(a)) \geq R_{\min};
\]

\[
E(R(a)) \geq A_{\min};
\]

Where \( a \) is a service selection, \( E(R(a)) \) represents the expected value of the random value \( R(a) \), \( R_{\min} \) \( A_{\min} \) is the minimum average reliability and availability respectively.
2.2 The analysis of composition structure

The quality of composition service depends not only on the structure of the service itself, but also on the composition structure. This study, based on previous work, gives the following some basic assembly structure, and analyzes their non-functional properties respectively.

1) Sequence pattern is an execution model in which services are carried out one by one, as Fig.1 shows. Since the services implement separately, the total expense for both executions is the sum of each service’s cost, the response time is computed in the same way. The reliability and availability can be calculated by multiplying each value respectively.

![Figure 1. Sequence pattern](image1)

Price: \( P(a_1) + P(a_2) \); Response time: \( T(a_1) + T(a_2) \);
Availability: \( A(a_1) \times A(a_2) \); Reliability: \( R(a_1) \times R(a_2) \);

2) Loop pattern depicts a particular sequential process where executes the same service repeatedly, as Fig. 2 illustrates. The aggregative QoS effects depend on the number of repeat executions. The calculation of the QoS is similar with sequence pattern.

![Figure 2. Loop pattern](image2)

Price: \( n \times P(a_1) \); Response time: \( n \times T(a_1) \);
Availability: \( A(a_1)^n \); Reliability: \( R(a_1)^n \);

3) Synchronization is that two or more activities execute in any order or in parallel. It does not proceed with the execution of the following activities until all these preceding activities have completed.

![Figure 3. Synchronization pattern](image3)

Price: \( \sum_{n=1}^{5} P(a_n) \);
Response time: \( T(a_1) + \max(T(a_2), T(a_3)) + T(a_3) + T(a_4) + T(a_5) \);
Availability: \( A(a_1) \times \ldots \times A(a_5) \);
Reliability: \( R(a_1) \times \ldots \times R(a_5) \);

4) Exclusive choice pattern is that one execution path from many alternatives is performed after the completion of the former service. Parallel services carry out independently, the follow-up service performs after only one alternative service.
Price: \[ \sum_{n=1}^{4} P(a_n) \];
Response time: \( T(a_1) + q_1 T(a_2) + q_2 T(a_3) + T(a_4) \);
Availability: \( A(a_1) \cdot (q_1 A(a_2) + q_2 A(a_3)) \cdot A(a_4) \);
Reliability: \( R(a_1) \cdot (q_1 R(a_2) + q_2 R(a_3)) \cdot R(a_4) \);

The objective of optimal service selection is that the selected services will satisfy the service requester’s requirements. Actually, the implementation models are very complicated which include a variety of basic patterns. The premise of properly evaluating the QoS performance is a reasonable assessment structure. Due to the complexity of the implementation models, the pattern’s worst case should only be considered.

3. Algorithm

Many former research works use the naive and inefficient way which would be to generate all possible service selections. In this paper, we propose an optimal algorithm which does not require one to generate the entire solution space. In new algorithm, availability and reliability constraints based on expected value are put forward. The selected service which violates the constraints will be removed from the solution space.

3.1 Expected value

In probability theory and statistics, the expected value of a random variable is the integral of the random variable with respect to its probability measure, It is often helpful to interpret the expected value of a random variable as the long-run average value of the variable over many independent repetitions of an experiment.

If the probability distribution of X admits a probability density function \( f(x) \), then the expected value can be computed as:

\[ E(X) = \int_{-\infty}^{\infty} x f(x) \, dx \] .

The computation of the expected value can be done using the property that the expected value of a linear combination of random variables is a linear combination of the expected values of these random variables. Thus,

\[ E(q_1X_1 + ... + q_nX_n) = \sum_{i=1}^{n} q_i E(X_i) \] , (2)

In general, the expected value operator is not multiplicative, i.e. \( E(XY) \) is not necessarily equal to \( E(X)E(Y) \). But if \( X \) and \( Y \) variables are independent, we can obtain:

\[ E(XY) = E(X)E(Y) \] , (3)

We take exclusive choice pattern as an example. Applying (2), (3) and (4), the expected value of reliability can be gotten as follow:
\[ E(R) = E(R_1) \times (q_1 E(R_2) + q_2 E(R_1)) \]

Where \( q_1, q_2 \) is the choice probability, and \( q_1 + q_2 = 1.0 \leq q_1, q_2 \leq 1 \). Since \( 0 < R(a_i) < 1 \), we can know \( 0 < E(R_i) < 1 \), \( 1 \leq i \leq 4 \). The following inequality can be derived:

\[
R_{\text{min}} \leq E(R) \leq E(R_1)
\]

\[
R_{\text{min}} \leq E(R) \leq E(R_1) \times (q_1 E(R_2))
\]

\[
R_{\text{min}} \leq E(R) \leq E(R_1) \times (q_1 E(R_2) + q_2 E(R_3))
\]

\[
R_{\text{min}} \leq E(R) \leq E(R_1) \times (q_1 E(R_2) + q_2 E(R_3)) \times E(R_4)
\]

The above inequality is the reliability constraints. The calculation of availability is similar with reliability. Each subset of the solution should satisfy the constraints when the QoS performance is being evaluated.

### 3.2 Computation of objective function

According to 1, the objective is to find the composition which has the minimum of four qualities. As the computing of reliability and availability in many patterns are nonlinear, it is not convenient to dynamically calculate the objective function. To easily solve the problem, we apply logarithm to transform reliability and availability into the linear form, as shown below. Because the logarithm does not alter the monotonic of function, that is to, a higher value still indicates better quality after the convert.

\[
A^* = \log(A(a_1) \times \ldots \times A(a_n)) = \log(A(a_1)) + \ldots + \log(A(a_n))
\]

\[
R^* = \log(R(a_1) \times \ldots \times R(a_n)) = \log(R(a_1)) + \ldots + \log(R(a_n))
\]

We take exclusive choice pattern as an example, as shown below. The objective is to determine the best service portfolio by finding out the minimum value of the objective function. The expression is rearranged by task in order to use the previous results when add a new task.

\[
\min F(a) = \min(T(a) + P(a) + R(a) + A(a))
\]

\[
= \min(\sum_{n=1}^{4} P(a_n) + T(a_1) + q_1 T(a_2) + q_2 T(a_3) + T(a_4) + \log(A(a_1)) + \log(q_1 A(a_2) + q_2 A(a_3)) + \log(A(a_4))
\]

\[
+ \log(R(a_1)) + \log(q_1 R(a_2) + q_2 R(a_3)) + \log(R(a_4)))
\]

\[
\geq \min(P(a_1) + T(a_1) + \log(A(a_1))) + \log(R(a_1)) + \ldots + P(a_4) + T(a_4) + \log(A(a_4)) + \log(R(a_4)))
\]

### 3.3 The new algorithm

This algorithm receives as input an expression for the reliability and availability, and analyzes partial selections of service providers. For each sub-selection, the reliability and availability of selected service are compared against their respective constraints. If both of the constraints are violated, the service will be removed from the solution space. Then we evaluate the overall quality of the partial selection after adding the new service, if the constraints are not satisfied, the selection is dropped and all other selections that have the same partial selection will not even be considered.

The proposed mathematical programming based approach is as follow:

Objective: \( \min(\sum_{i=1}^{n} (P(a_i) + T(a_i) + \log(R(a_i)) + \log(A(a_i)))) \)

Input: \( s_{ij} \) is the \( j^{th} \) candidate service of task \( i \);

\( R_{\text{min}}, A_{\text{min}} \) are the preset minimum of the reliability and availability respectively.
Output: the optimal sequence of Web service composition.

Function OptimalSelection()

\[ i = 1 ; \] //initialize task pointers

\[ z_{opt} = s_{11} ; \] // initialize the optimal selection

\[ z = s_{11} ; \] // initialize the solution

While \( i \leq n \)

For \( j = 1 \) to \( N_i \)

If \( E(R(s_{ij})) \leq R_{min} \) or \( E(A(s_{ij})) \leq A_{min} \) then

Delete \( s_{ij} \);

\[ N_i = N_i - 1 ; \]

Break;

End if

\[ z = z_{temp} + s_{ij} ; \]

If \( E(R(z)) \geq R_{min} \) and \( E(A(z)) \geq A_{min} \) then

If \( F(z) \leq F(z_{opt}) \) then

\[ z_{opt} = z ; \]

End if;

End if;

End for;

\[ z_{temp} = z_{opt} ; \]

End while;

Return \( z_{opt} ; \)

4. Simple example

In this section our algorithm is evaluated and compared to the general algorithm with regard to the computation time and the excellence in approximating the optimal solution. The general method traverses all Web service and matches input and output of Web service to generate service composition. The new algorithm removes the service composition in time which does not satisfy the conditions to reduce computing time.

The experiment is carried out in a PC with Windows XP SP2, where the CPU is Intel Pentium 3GHz and the Memory is 2GB and the development platform is matlab. Based on the business process illustrated in Fig. 5, this study randomly generates services for each task, and chooses eight data set of which service average is 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000. In our example, with constraints \( R_{min} = 0.9 \) and \( A_{min} = 0.9 \). The choice probability of task \( a_5 \) and \( a_6 \) is \( q_1 = 0.4 \), \( q_2 = 0.6 \). The cycle number of \( a_4 \) is 10.
Even though there are numerous candidate services for each task, the following section shows that the performance of the proposed mathematical programming based solution is still very efficient. In Fig.6, curves of the time versus the number of services for each task are plotted. As shown in the figure, the time achieved by our proposed algorithm is shorter than the general one. Especially, the time can be saved by about one third at 8000 services which indicates the performance achieved by our proposed algorithm is superior especially when the number of services is large.

5. Conclusion

Service composition brings several new benefits to software design and architecture by enabling re-use and sharing of components through dynamic discovery. Due to the distributed and dynamic nature of service computing, the adoption of Web service may suffer from several uncertainties. Thus, it’s important to make sure that service is always available and reliable. In this work, we want to find an allocation that minimizes the execution time and cost of the business process subject to reliability and availability constraints based on expected value. First, we identify the impact of various structural aspects of the composition in terms of the performance and outcomes of the composition. Then, an algorithm is proposed which can reduce the computing time and makes sure better quality of the services selection at the same
time by examining a very tiny fraction of the solution space. Finally, we compare the time obtained by our proposed algorithm with the one achieved by other algorithm, and prove the advantage of the proposed algorithm.

References


