World Wide Web (2012) 15:663–684
DOI 10.1007/s11280-012-0157-5

A novel QoS model and computation framework in web service selection

Yanan Hao · Yanchun Zhang · Jinli Cao

Received: 2 April 2011 / Revised: 12 January 2012 /
Accepted: 18 January 2012 / Published online: 28 February 2012
© Springer Science+Business Media, LLC 2012

Abstract With the rapid development of e-commerce over Internet, web services have attracted much attention in recent years. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web. Then they can dynamically combine individual services to provide new value-added services. With the increasing number of web services having equivalent functionality, the binding procedure is driven by some non-functional, Quality of Service (QoS) criteria, such as the money cost, response time, reputation, reliability or a trade-off between them. Thus, an important problem is, given QoS constraints, how to aggregate and leverage individual service's QoS information to derive the optimal QoS of the composite service. In this paper, we propose a novel QoS model for performing flexible service selection. The key idea of the model is to relax users' QoS constraints and try to find the most possible services satisfying users' QoS requirements. Based on the proposed QoS framework, we develop various algorithms for making service selection on individual and composite services. We also introduce a top-k ranking strategy to reflect a user's personalized requirements. Experimental evaluation shows the proposed QoS model is efficient and practical.

Keywords web service · QoS

Y. Hao (✉) · Y. Zhang
Center for Applied Informatics, Victoria University, Melbourne, Australia
e-mail: yanan.hao1@live.vu.edu.au

Y. Zhang
e-mail: yanchun.zhang@vu.edu.au

J. Cao
Department of Computer Science and Computer Engineering, La Trobe University, Bundoora, VIC 3086, Australia
e-mail: j.cao@latrobe.edu.au

1 Introduction

A web service is programmatically available application logic exposed over Internet. It has a set of operations and data types. The current set of web service specifications defines how to specify reusable operations through the Web-Service Description Language(WSDL), how these operations can be discovered and reused through the Universal Description, Discovery, and Integration(UDDI) API, and how the requests to and responses from web-service operations can be transmitted through the Simple Object Access Protocol API(SOAP). With the rapid development of e-commerce over Internet, web services have attracted much attention in recent years. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web. Then they can combine individual services into more complex, orchestrated services.

Recently, the process-based approach to web service composition has gained considerable momentum and standardization [1]. In this scenario, a service composition can be regarded as a process model containing abstract service specifications, without specifying actual services needed to be bound to the process model. With the increasing number of web services having equivalent functionality, the binding procedure is driven by some non-functional, Quality of Service (QoS) criteria, such as the money cost, response time, reputation, reliability or a trade-off between them [6]. Thus, an important problem is, given QoS constraints, how to aggregate and leverage individual services' QoS information to derive the optimal QoS of the composite service. Actually, dynamic binding of service compositions constitutes one of the most interesting challenges for service-oriented architectures.

A lot of methods have been proposed to solve this problem, including linear programming [28], reduction rules method [8], utility function strategy [26], CP-nets [23] and AND/OR graph method [21], etc. However, most approaches only focus on obtaining optimized service selection under user requirements and overlook user preferences. As an example, consider a user who is searching for the *postcode* of a given city. Table 1 gives a collection of candidate web services with the same function. All these services can retrieve the postcode of a given city, although they have different non-functional properties, e.g. *Pri.*(i.e. price($)), *Res.*(i.e. response time(s)), *Rep.*(i.e. reputation score(points/100)) and *Rel.*(i.e. reliability probability). A typical form of QoS query p issued by the user is $M\theta v$, where M is a QoS metric, v is a constant value, and θ is a comparison operator such as $=, <, >, \leq$ *or* $\geq$. Examples are p_1: $price < \$10$, p_2: $response\ time < 11s$, p_3: $reputation\ score \geq 70$ and p_4: $reliable\ probability > 0.5$, etc. These conditions can also be combined by an "AND" operator, denoted as $price < \$10$ AND $response\ time < 11s$ AND $reputation \geq 70$ AND $reliability > 0.5$, forming a QoS query vector $p = (p_1, p_2, p_3, p_4)$. In order to fulfil the user's request, existing methods need to seek services satisfying all the four

Table 1 Sample web services.

Web services	Quality of Service (QoS)			
	Pri.	Res.	Rep.	Rel.
s_1	26	5	88	0.1
s_2	14	10	70	0.9
s_3	35	9	36	0.3
s_4	5	1	90	0.6

Each web service provides the same postcode information.

conditions above simultaneously and then return them. However, several problems may arise when applying these techniques: firstly, these methods can not solve the *empty result* problem. Since the user issues his requirements without knowing all the detailed QoS properties of all available services, maybe there are no fully satisfying services, i.e. services satisfying all the conditions at one time. In such cases, obviously relaxing the user's request and returning some approximate results is a much better idea than just reporting an empty result; secondly, these methods lack flexibility and the personalization problem has been overlooked. Depending on their context, different users may have different preferences about the services they need. For instance, a user may say service s_2 is better than service s_4, because although s_2 is more expensive than s_4 and exceeds the user's financial requirement a bit, it has much better reliability, which is crucial to her because of her strict reliability requirements. Similarly, another user running out of money prefers a cheaper service, while a cautious user may prefer services with "excellent" reputation, etc. So, whether a service is good or not in QoS depends on the user's context and preference. It is essential to model the user's personalized preferences and requests.

In this paper, we present novel techniques to solve the challenges above, and give experimental evidence that shows our methods are efficient and practical. In particular, the main contributions of this paper are listed as follows:

1. We propose a novel QoS evaluation model for performing flexible and adaptable service selection from the point of view of users. The key idea of the model is to relax the user's QoS constraints and try to find the most possible services which meet the user's QoS requirements.
2. Based on the proposed QoS framework, we develop various algorithms for making service selection on individual and composite services, respectively. We also introduce a top-k ranking strategy to reflect a user's personalized requirements.
3. We present the experimental result of a thorough evaluation. Experimental evaluation shows the proposed QoS model is efficient and practical.

2 QoS computing model

In this section, we formulate the QoS computing model for individual services and composite services, respectively.

2.1 Computing model for individual services

2.1.1 QoS metrics

Many QoS metrics have been proposed in the literature. Typical criteria include response time, throughput, price, reputation, reliability, transactional properties and security, etc. In this paper, we only consider four basic metrics, which are almost available for all web services [15]: price, duration, reputation and reliability.

1. **Price**. This is the amount of money that a service requester has to pay for executing a task. The task may be either an element web service or a composite web service. Users can get execution price via online advertisement or inquiry

entry available through service providers. Given a service s, we use $price(s)$ to denote the price for executing s.

2. **Duration**. The execution duration measures the response time from the submission of a request to the receiving of the response. The average response time can be estimated based on observation of past executions. Given a service s, we use $duration(s)$ to denote its execution duration.
3. **Reputation**. The reputation of a service s reflects its trustworthiness. It mainly depends on usage evaluation carried out by end users. Usually, users are given a range, for example from 1 point to 100 points, to rank a web service. We define the reputation of service s as the average ranking given by users, denoted as $reputation(s)$.
4. **Reliability**. The reliability of a service is the probability that the service can successfully complete within a time limit. Its value is computed from past invocations history. We use $reliability(s)$ to describe the reliability of a service s.

Given the QoS metrics above, the quality of a web service s is defined as a four-dimensional vector: $q(s) = (price(s), duration(s), reputation(s), reliability(s))$. Let $S = \{s_1, s_2, ..., s_n\}$ be a group of web services with the same functional properties. For each service $s_i \in S$, its quality vector is represented as $q(s_i) = (q_i(1), q_i(2), q_i(3), q_i(4))$, corresponding to price, duration, reputation and reliability in order. Given a user's QoS constraints, our algorithms aim to determine the best services in S satisfying the user's requirement. It is worth noting that although the number of metrics criteria is limited in this paper, our model is extensible. New metric factors can be easily added without fundamentally changing the algorithms for QoS computation. As the quality of a web service s is represented as a four-dimensional vector, we will use *dimension* and *metric* interchangeably in the remainder of the paper.

2.1.2 Normalization of QoS values

The QoS metrics proposed in Section 2.1.1 are not consistent with each other. As we can see, the higher the value of price or duration is, the lower the quality; whereas the higher the reputation or reliability is, the better the quality is. In order to provide a uniform representation of QoS metrics, we need to normalize QoS values before we do QoS computation. Suppose we have the quality vectors for all services $\in S$. For each service s_i's quality vector $q(s_i) = (q_i(1), q_i(2), q_i(3), q_i(4))(1 \leq i \leq n)$, we normalize its values to obtain $q'(s_i) = (q'_i(1), q'_i(2), q'_i(3), q'_i(4))$, where

$$q'_i(j) = \frac{q^j_{\max} - q_i(j)}{q^j_{\max} - q^j_{\min}},\ j = price(1), duration(2) \tag{1}$$

$$q'_i(j) = \frac{q_i(j) - q^j_{\min}}{q^j_{\max} - q^j_{\min}},\ j = reputation(3), reliability(4) \tag{2}$$

In the equations above, we can safely assume $q^j_{\max} \neq q^j_{\min}$, where $q_i(j)$ is the QoS of s_i on metric j, $q^j_{\max} = Max\{q_i(j)\}, 1 \leq i \leq n$ and $q^j_{\min} = Min\{q_i(j)\}, 1 \leq i \leq n$. When a user needs to find services satisfying his or her quality requirements, for consistency, it is also needed to include the QoS query vector p in the normalization process, so

we have $q^j_{\max} = Max\{\{q_i(j)|i = 1, 2, ..\} \cup \{p_j\}\}, 1 \le i \le n$ and $q^j_{\min} = Min\{\{q_i(j)|i = 1, 2, ..\} \cup \{p_j\}\}, 1 \le i \le n$.

Example 1 Suppose there are four web services in S and that their values of QoS metrics are given in Table 1. We also assume that a user's QoS query vector is $p = (p_1, p_2, p_3, p_4)$, where p_1 is *price* $< \$10$, p_2 is *response time* $< 8s$, p_3 is *reputation* > 50, and p_4 is *reliability* > 0.7. Using (1) and (2), we have the normalized Table 2.

After normalization, the quality on each metric is monotonic increasing with its corresponding metric value, i.e. the greater the QoS value on one metric is, the better the quality of the service on the metric. So, users only need to issue a QoS query vector using one form: $M > v$, where M is a QoS metric and v is a constant value related to M. For instance, the query vector p in example 1 can be converted into $p' = (p'_1, p'_2, p'_3, p'_4)$, where p'_1 is *price* > 0.8, p'_2 is *response time* > 0.2, p'_3 is *reputation* > 0.3, and p'_4 is *reliability* > 0.8.

2.1.3 Answering a user's QoS query

In this section, we focus on how to return desirable services from a group of services S with the same function, given a QoS query. Before giving the definition of query result, we first define the quality difference of a user's QoS query p with regard to a service s_i's quality vector $q(s_i)$ as follows:

Definition 1 Suppose $p = (p_1, p_2, p_3, p_4)$ is a four-dimensional QoS query vector, such that p_k is of the form of $M_k > v_k$,where $k \in \{1, 2, 3, 4\}$, $M_k \in \{price, duration, reputation, reliability\}$, and v_k is the QoS value of M_k. $q(s_i) = (q_i(1), q_i(2), q_i(3), q_i(4))$ is an individual service s_i's quality vector, where $q_i(1)$, $q_i(2)$, $q_i(3)$ and $q_i(4)$ is the QoS value on price, duration, reputation and reliability, respectively. Let $\delta_{ik} = v_k - q_i(k)(k = 1, 2, 3, 4)$. We say $D(p, q(s_i)) = (\delta_{i1}, \delta_{i2}, \delta_{i3}, \delta_{i4})$ is the *individual quality difference* between the user's QoS query vector p and the individual service s_i's quality vector $q(s_i)$.

Based on the monotonic increasing property of each QoS metric, it is straightforward to draw the following property:

Property 1 *The less δ_{ik} is, the better the quality of service s_i on metric k.*

In order to describe the quality of service s_i relative to a query vector p, by Property 1, we say that s_i satisfies p strictly if $\forall k \in \{1, 2, 3, 4\}, \delta_{ik} \le 0$. s_i is said not to satisfy p strictly if $\exists k \in \{1, 2, 3, 4\}, \delta_{ik} > 0$. Generally, $\forall i \ne j, D(p, q(s_i)) \ne D(p, q(s_j))$,

Table 2 Normalization of QoS values of services in Table 1.

Web services	Quality of Service (QoS)			
	Pri.	Res.	Rep.	Rel.
p	> 0.8	> 0.2	> 0.3	> 0.8
s_1	0.3	0.6	0.9	0
s_2	0.7	0	0.6	1
s_3	0	0.1	0	0.3
s_4	1	1	1	0.6

which means service s_i may be better or worse than s_j on some metrics. But, how can we compare these two services as a whole? To formally describe this kind of relationship, we employ dominance checking between the *individual quality difference* of p with regard to different services in S.

Definition 2 Given a QoS query vector p and two services $s_i, s_j \in S$, if the value of $D(p, q(s_i))$ on each dimension is not larger than that of $D(p, q(s_j))$ and strictly smaller on at least one dimension, then we say service s_i dominates service s_j with respect to the query vector p, denoted as $s_i \succ s_j$. In other words, we can say service s_i is better than service s_j.

For example, consider the running example in Table 2. According to definition 1, we have $D(p, q(s_1)) = (0.5, -0.4, -0.6, 0.8)$, $D(p, q(s_2)) = (0.1, 0.2, -0.3, -0.2)$, $D(p, q(s_3)) = (0.8, 0.1, 0.3, 0.5)$, and $D(p, q(s_4)) = (-0.2, -0.8, -0.7, 0.2)$. $D(p, q(s_4))$ is smaller than $D(p, q(s_3))$ and $D(p, q(s_1))$ on every dimension, but its fourth metric is larger than $D(p, q(s_2))$. Hence, $s_4 \succ s_1$ and $s_4 \succ s_3$.

Definition 3 The query result of a QoS query vector p on a group of services S, denoted as $R(p, S)$, is the set of all the services $\in S$, each of which is not dominated by any other service $\in S$.

In the example in Table 2, since $s_4 \succ s_1$ and $s_4 \succ s_3$, we can say s_1 and $s_3 \notin R(p, S)$. Also, s_2 is not dominated by any other service in S. Thus the query result of p on S is $R(p, S) = \{s_2, s_4\}$.

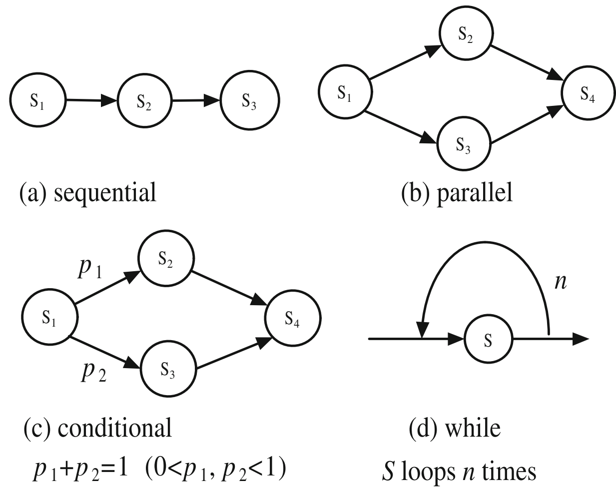
Furthermore, it is very common that different users may have different preferences for some metrics. We can specify a weight for each metric to adjust the query results, satisfying users' dynamic and personalized QoS requirements.

The goal of our task is to find QoS query results for individual services and composite services. However, as the number of available services increases, the cost of finding query results will be large. How to develop efficient methods to return satisfying results is the central problem. We will discuss various algorithms to achieve this in later sections. In the next section, we give the QoS computing model for composite services.

2.2 Computing model for composite services

A composite service is actually a business process integrating necessary individual services. So, the quality metrics for individual services are also applied to composite services. The QoS metric value of a composite service is determined by the QoS metric values of its individual services, and the workflow pattern capturing the control flow and dependencies between individual services. There are more than twenty different patterns [20] through which individual services can be integrated to form a composite service, but only four basic workflow patterns among them are essential: *sequential*, for defining an execution order; *parallel*, for parallel routing; *switch*, for conditional routing; and *while*, for looping. Figure 1 shows these four basic workflow patterns. Each pattern contains some nodes, which are also called tasks in this paper. We deal with the sequential workflow pattern first. Later we will see its computing methods with QoS can also be applied to the other three patterns.

Figure 1 Workflow patterns.



2.2.1 Sequential composition model

Now let us see how to compute the QoS metric values of a sequential composite service s with the running pattern like Figure 1a. Using the same quality metrics as individual services, the aggregation functions for s are given as below:

1. **Price**. The price of s is the sum of each of its individual service s_i's price. Formally, we have $price(s) = \sum_{i=1}^{n} price(s_i)$.
2. **Duration**. The duration of s is the sum of each of its individual service s_i's duration; that is $duration(s) = \sum_{i=1}^{n} duration(s_i)$.
3. **Reputation**. The reputation of s is the average of each of its individual service s_i's reputation point, denoted as $reputation(s) = (\sum_{i=1}^{n} reputation(s_i))/n$. In this paper, we map $reputation(s) \times n$ to $reputation'(s)$, thus the function above becomes $reputation'(s) = \sum_{i=1}^{n} reputation(s_i)$.
4. **Reliability**. The reliability of s is the probability product of each of its individual service s_i's reliability probability: $reliability(s) = \prod_{i=1}^{n} reliability(s_i)$. We use the logarithmic function $\ln(x)$ to linearize this function. Let $\ln reliability(s) = \ln \prod_{i=1}^{n} reliability(s_i)$, and $reliability'(s_i) = \ln reliability(s_i)$, then we have a linear aggregation function to compute the reliability of s: $reliability'(s) = \sum_{i=1}^{n} reliability'(s_i)$.

Using the aggregation functions above, the quality of a sequential composite service s can be unified as $q(s) = \sum_{i=1}^{n} q(s_i)$. Here s and s_i have been mapped and linearized.

2.2.2 General composition model

Besides sequential structures, real-world composite services often have loop operations, conditional operations, and parallel operations to run services simultaneously. These operations can be converted into sequential model according to a group of rules similar to [12, 27] and [28]:

1. We unfold a loop operation to a sequential structure by cloning the cyclic nodes n times, where n is the looping count of the loop structure.
2. Both conditional and parallel operations contain multiple branches. Each of the branches can be regarded as an independent sequential operation.
   - (a) For a conditional operation, since only one branch is executed with a probability p at runtime, we calculate its QoS value by averaging the QoS values of all its branches.
   - (b) In parallel structures, all branches are executed simultaneously at runtime, so we need to combine the QoS values of all branches. For price, reputation and reliability metrics, all branch QoS values are summed up as the overall QoS of the parallel operation; whereas for execution duration, the overall QoS value is defined as the maximum QoS value of all branches on this metric.

As an example, Figure 2 gives a composite service E with the four workflow patterns above. S_1 is followed by either S_2 or S_3 with a probability of p_1 or p_2; S_5 is followed by both S_6 and S_7 in a parallel way. S_6 is iterated for at most n times. Using the same notations as the sequential model, the overall *duration* and *price* of E are formulated as follows:

$$\begin{aligned} dur(E) &= dur(s_1) + p_1 * dur(s_2) + p_2 * dur(s_3) + dur(s_4) \\ &\quad + dur(s_5) + \max\left(dur(s_6) * n, dur(s_7)\right) + dur(s_8) \end{aligned} \tag{3}$$

$$\begin{aligned} pri(E) &= pri(s_1) + p_1 * pri(s_2) + p_2 * pri(s_3) \\ &\quad + \sum_{i=4,5,7,8} pri(s_i) + pri(s_6) * n \end{aligned} \tag{4}$$

Similar to *price*, the QoS value on *reliability* and *reputation* can be derived the same way. We omit their equations here due to space limit.

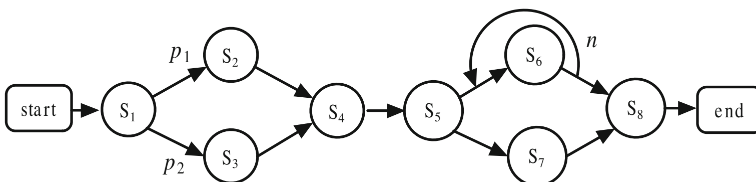


Figure 2 Composite service example.

2.2.3 Normalization

As we have seen in individual services, in order to provide consistent QoS metrics for composite services, we need to normalize each of its individual service's QoS value before we do QoS computation, except that this time the normalization is within all candidate groups of services instead of a single group. Suppose we have a composite service C with k tasks in its execution flow. Each task is executed by a service $S_{ij}(i = 1, 2, ..., k;\ j = 1, 2, ...)$ from service group $S_i(i = 1, 2, ..., k)$, which consists of $|S_i|$ services having the same functionality as S_{ij}. For each service $S_{ij} \in S_i$, its quality vector $q(S_{ij})$ is normalized by the two equations below:

$$q'_{ij}(m) = \frac{q^m_{\max} - q_{ij}(m)}{q^m_{\max} - q^m_{\min}}, m = price, duration \tag{5}$$

$$q'_{ij}(m) = \frac{q_{ij}(m) - q^m_{\min}}{q^m_{\max} - q^m_{\min}}, m = reputation, reliability \tag{6}$$

where $q_{ij}(m)$ is the quality value of service S_{ij} on metric m, $q^m_{\max} = Max\{q_{ij}(m)\}$, $1 \leq i \leq k$, $j = 0, 1, ..., |S_i|$ and $q^m_{\min} = Min\{q_{ij}(m)\}$, $1 \leq i \leq k$, $j = 0, 1, ..., |S_i|$.

Similar to individual services, we also include the user's QoS query vector in the normalization process. Before formally defining the QoS Query on composite services, we give a running example first.

Example 2 Figure 1a shows a composite service with three sequential functions. Each function is executed by an individual service S_{ij} from service group $S_i(i = 1, 2, 3)$. The QoS metric values of all individual services in each service group are shown in Table 3. Also, the QoS query vector for this composite service is given at the first row, i.e. $p = (p_1, p_2, p_3, p_4)$, where p_1 is *price* > 0.8, p_2 is *response time* > 0.2, p_3 is *reputation* > 0.3, and p_4 is *reliability* > 0.8. Without loss of generality, we assume all these values have been normalized and linearized, as shown in Section 2.2.1.

2.2.4 Answering a user's QoS query on composite services

Following the definition of QoS query result for individual services, in this section we discuss how to return desirable services contributing to a composite service. In order to simplify the problem, we need to give several definitions first.

Table 3 Example of composite service.

Service group	Individual service	Quality of Service (QoS)			
		Pri.	Res.	Rep.	Rel.
p		> 0.8	> 0.2	> 0.3	> 0.8
S_1	s_{11}	0.8	0.2	0.3	0.8
S_1	s_{12}	0.3	0.6	0.9	0
S_2	s_{21}	0.7	0	0.6	1
S_2	s_{22}	0	0.1	0	0.3
S_3	s_{31}	1	1	1	0.6
S_3	s_{32}	0.9	0.1	0.2	0.7
S_3	s_{33}	0.4	0.3	0.5	0.2

Definition 4 We assume C_i is a process with k tasks, each of which is executed by a candidate service S_i^{ti} from service group $S_i(i = 1, 2, ..., k)$. Then $C_i = \{S_1^{t1}, S_2^{t2}, ..., S_i^{ti}, ..., S_k^{tk}\}$ is called a *composite service*. We use C to denote the composite service set containing all composite services.

Definition 5 Given a QoS query vector $p = (p_1, p_2, p_3, p_4)$, where p_k is of the form of $M_k > v_k, k \in \{1, 2, 3, 4\}$, $M_k \in \{price, duration, reputation, reliability\}$ and v_k is the QoS value of M_k; and given the quality vector $q(C_i) = (q_i(1), q_i(2), q_i(3), q_i(4))$ of a composite service C_i, where $q_i(1)$, $q_i(2)$, $q_i(3)$ and $q_i(4)$ is the QoS value of C_i on price, duration, reputation and reliability, respectively. Let $\delta_{ik} = v_k - q_i(k)(k = 1, 2, 3, 4)$. We say $D(p, q(C_i)) = (\delta_{i1}, \delta_{i2}, \delta_{i3}, \delta_{i4})$ is the *composite quality difference* between the user's QoS query vector p and the composite service C_i's quality vector $q(C_i)$.

Clearly, according to the definition and the aggregation functions proposed in Section 2.2.1, we can conclude the composite quality difference between a user's QoS query vector p and a sequential composite service C_i's quality vector $q(C_i)$ can be calculated as the sum of all the individual quality difference of p with regard to each candidate service contributing to C_i. We formulate this as

$$D(p, q(C_i)) = \sum_{i=1}^{k} D\left(p, S_i^{ti}\right) \tag{7}$$

where S_i^{ti} is the service from service group S_i contributing to C_i.

Similar to individual services, the dominance relationship between composite services is defined below:

Definition 6 Given a QoS query vector p and two services C_i, C_j, if the value of $D(p, q(C_i))$ on each dimension is not larger than that of $D(p, q(C_j))$ and strictly smaller on at least one dimension, then we say service C_i dominates service C_j with respect to the query vector p, denoted as $C_i \succ C_j$.

For example, in Table 3, consider a QoS query vector $p = \{0.8, 0.2, 0.3, 0.8\}$ and two sequential composite services $C_1 = \{s_{11}, s_{21}, s_{31}\}$, $C_2 = \{s_{12}, s_{22}, s_{33}\}$. By definition 5, we have $D(p, q(C_1)) = D(p, q(s_{11})) + D(p, q(s_{21})) + D(p, q(s_{31})) = \{-0.1, -0.6, -1, 0\}$, and $D(p, q(C_2)) = D(p, q(s_{12})) + D(p, q(s_{22})) + D(p, q(s_{33})) = \{1.7, 0.5, -0.5, 1.9\}$. $D(p, q(C_1))$ is smaller than $D(p, q(C_2))$ on every dimension, so $C_1 \succ C_2$. In other words, we can say C_1 is better than C_2 for satisfying the query vector p.

Following the same way as definition 3, the computing model for composite services is shown as follows:

Definition 7 The query result of a QoS query vector p on a composite service set C, denoted as $R(p, C)$, is the set of all the composite services $\in C$, each of which is not dominated by any other composite service $\in C$.

In the example of Table 3, there are a total of 12 candidate composite services, each with different composite quality difference with regard to query vector p. The

number of possibilities goes up exponentially with an increasing number of service groups. So, developing efficient algorithms for computing query result of p on a composite service set is particularly challenging in our case. In this paper, we design various algorithms to achieve this goal.

3 Algorithms for answering a user's QoS query

In this section, we present different approaches of answering a user's QoS query on individual services and composite services, respectively.

3.1 Algorithms for QoS query on individual services

Recall the model we defined in Section 2.1.3. Our goal is to compute the query result of a QoS query vector p on a group of services S, $R(p, S)$, i.e., the set of all the services $\in S$, each of which is not dominated by any other service $\in S$.

3.1.1 A naïve approach

A naïve strategy for the QoS query is to compute the individual quality difference of every service in S and then, for each quality difference, make dominance checking with all the other quality differences to find the satisfying services. However, the naïve strategy is expensive since every two quality differences are compared. If we consider the rapid increase of available services, the cost of the naïve approach would be even more expensive.

3.1.2 The RC algorithm

Algorithm 1 The RC algorithm for QoS query on individual services

```
input  : A QoS query vector p, A group of services S
output: A query result set R containing satisfying services
R = φ;
foreach service s ∈ S do
    if there exists a service s′ ∈ R such that s′ ≻ s with respect to p then
        discard s;
    else
        insert s to R;
        remove every service s′ ∈ R such that s ≻ s′ with respect to p;
    end
end
return R;
```

The main cost the naïve approach is the duplicate comparison between quality differences. This algorithm, called ***R**educe **C**omparison* (RC) approach, tries to reduce the number of comparisons by using the comparing result already available. It traverses all the services in S and maintains a set R to keep the satisfying services obtained so far. For each service s, the RC algorithm checks if s is dominated by a service in R with respect to the query vector p. We discard s if it is dominated; otherwise we insert it to R and discard those services in R dominated by s. When all

services in S have been traversed, the RC algorithm ends and returns R as the QoS query result. Detailed steps of the RC algorithm are presented in Algorithm 1.

3.1.3 R-tree based RC algorithm (RRC)

The RC algorithm outperforms the naïve approach, as it avoids duplicate comparisons. However, the performance of RC algorithm may decrease as the number of services in R increases. To overcome the problem, the R-tree based RC algorithm (RRC) is proposed to facilitate the retrieval of services. Following most methods in the relevant references, we index the metric values of each quality vector corresponding to each service in S. An intermediate entry E_i corresponds to the *minimum bounding rectangle* (MBR) of a node N_i at the lower level, while a leaf entry corresponds to a quality vector. The dominance relationship between services can be easily determined by bound check similar to BBS algorithm in [17].

In the RRC algorithm, the *maximum quality difference* between p and a MBR, say E (i.e., intermediate entry), denoted as $D_{\max}(p, q(E))$, equals to the quality distance of its lower bound (e.g. the lower-left corner point in two dimensional space) quality vector with respect to p, while the *minimum quality difference* between p and E, denoted as $D_{\min}(p, q(E))$, equals to the quality distance of its upper bound (e.g. the higher-right corner point in two dimension space) quality vector with respect to p. We say a service s dominates a MBR E, if the following equation holds: $D(p, q(s)) \leq D_{\min}(p, q(E))$, and the inequality is strict on at least one QoS metric, as in this case service s must dominate all services in E.

Algorithm 2 The RRC algorithm for QoS query on individual services

```
input  : A QoS query vector p, an R-tree T indexing a group of services S
output: A query result set R containing satisfying services

R = φ;
insert into the stack all entries of the root node of T;
while stack not empty do
    remove top entry E;
    if there exists a service s′ ∈ R such that s′ ≻ E with respect to p then
        discard E;
    else
        if E is an intermediate entry then
            foreach child e_i of E do
                if e_i is not dominated by a service s′ ∈ R with respect to p then
                    insert e_i into the stack;
                end
            end
        else
            insert E to R;
            // E is a leaf entry, i.e. the quality vector of an individual
            service;
        end
    end
end
return R;
```

Based on the explanations above, we show how the algorithm works: it starts with the root node of the R-tree and inserts all its entries in a stack. If an entry is not dominated by the query quality vector p, then we expand it. That is to say it is removed from the stack and all its children are inserted into the stack. This process repeats until the stack is empty. Detailed steps of RRC algorithm are presented in Algorithm 2. As our experiments verified, the RRC algorithm is efficient and its performance is much better than RC.

3.2 Algorithms for QoS query on composite services

In this section, we propose methods to process the problem of QoS query on composite services. Recall Definition 7 in Section 2.2.4. Now the goal is to compute the query result of a QoS query vector p on a composite service set C, denoted as $R(p, C)$, i.e., the set of all the composite services $\in C$, each of which is not dominated by any other composite service $\in C$.

3.2.1 Straightforward strategy

Suppose each composite service $C_i \in C$ has k tasks in its execution flow. Each task is executed by a candidate service $S_i^{ti}(i = 1, 2, ..., k)$ belonging to service group $S_i(i = 1, 2, ..., k)$, the cardinality of which is $|S_i|$. A straightforward strategy for the QoS query on composite service set C is to enumerate all possible $C_i = \{S_1^{t1}, S_2^{t2}, ..., S_i^{ti}, ..., S_k^{tk}\}$ firstly and then compute their composite quality difference with respect to p, and finally return $R(p, C)$ by applying the proposed algorithms in Section 3.1 on C. Clearly, each service in S_i can contribute to C_i, so the total number of composite services in C is $\prod_{i=1}^{k} |S_i|$, which leads to much more expensive cost than QoS queries on individual services. If the number of services in service groups or the number of service groups increases, the cost of the straightforward strategy is even more expensive.

3.2.2 Locally pruning method

As we can see, the straightforward approach traverses all services in each service group then combines traversed individual services to find qualifying composite services, so the number of services in each service group is a key factor in the cost, and it would be useful if we can reduce the number of services in service groups. To achieve this, we need to identify and remove those services from each service group, which are impossible to appear in the final query result $R(p, C)$. In this section, the *locally pruning method*(LPM) is developed to address this issue. The intuition behind this approach is that a locally dominated service, i.e., it is dominated by some other service in the same group, does not appear in $R(p, C)$. This idea can be summarized by the following heuristic:

Heuristic 1 *Let S_i be a group of services and S_i^{ti} be an arbitrary service $\in S_i$. Service S_i^{ti} can be safely pruned if there exists a service $S_i^{ti'} \in S_i$ such that $S_i^{ti'} \succ S_i^{ti}$.*

Proof According to the four different workflow patterns in Figure 1, we give the proof separately.

Firstly, assume S_i^{ti} is selected to a sequential structure $C_i = \{S_1^{t1}, S_2^{t2}, ..., S_i^{ti}, ..., S_k^{tk}\}$ of a composite service in $R(p, C)$. Now we modify C_i by replacing S_i^{ti} with $S_i^{ti'}$ to obtain another sequential structure $C_i' = \{S_1^{t1}, S_2^{t2}, ..., S_i^{ti'}, ..., S_k^{tk}\}$. According to (7), we have $D(p, q(C_i)) = \sum_{l=1}^{k} D(p, S_l^{tl})$, and $D(p, q(C_i')) = \sum_{l=1}^{i-1} D(p, S_l^{tl}) + D(p, S_i^{ti'}) + \sum_{l=i+1}^{k} D(p, S_l^{tl})$.The dominance check between C_i and C_i' can be done using formula $D(p, q(C_i')) - D(p, q(C_i)) = D(p, S_i^{ti'}) - D(p, S_i^{ti})$. Note that $S_i^{ti'} \succ S_i^{ti}$, so the QoS value of $D(p, q(C_i')) - D(p, q(C_i))$ on each metric is not larger than zero and strictly smaller on at least one metric. That is to say, the value of $D(p, q(C_i'))$ on each dimension is not larger than that of $D(p, q(C_i))$ and strictly smaller on at least one dimension. Therefore, C_i' dominates C_i and C_i can not appear in $R(p, C)$, which is contradictory to the fact that $C_i \in R(p, C)$.

As for loop structures and conditional structures, as they can be converted into linear sequential structures according to the conversion rules in Section 2.2.2, the conclusion can be derived similarly.

Now we suppose S_i^{ti} is selected to a parallel structure C_i of a composite service in $R(p, C)$. Since C_i's QoS values on price, reliability and reputation can be accumulated, effectively it can be regarded as a sequential structure and the deduction above still applies. Therefore, we only need to prove the heuristic rule works on *duration* metric as well. Without loss of generality, we suppose C_i only contains two service groups S_i and S_j, where $S_i = \{S_i^{ti'}, S_i^{ti}\}$, $S_i^{ti'} \succ S_i^{ti}$ and $S_j = \{S_j^{tj}\}$. If $dur(S_j^{tj}) \leq dur(S_i^{ti'})$, then $dur(C_i)$ is determined by S_i^{ti} or $S_i^{ti'}$, so service S_i^{ti} can be safely pruned; if $dur(S_j^{tj}) \geq dur(S_i^{ti})$, then $dur(C_i)$ is determined by S_j^{tj}, thus clearly service S_i^{ti} can also be safely pruned; otherwise $dur(S_j^{tj})$ falls in between $dur(S_i^{ti'})$ and $dur(S_i^{ti})$ and we have the dominance relationship between two compositions: $\{S_i^{ti'}, S_j^{tj}\} \succ \{S_i^{ti}, S_j^{tj}\}$, therefore, service S_i^{ti} can be safely pruned, too. □

Algorithm 3 The LPM algorithm for QoS query on composite services

```
input  : A QoS query vector p
         k service groups S_1, S_2, ..., S_k
output: A query result set R containing satisfying composite services
R = φ;
foreach service service group S_i do
  | Locate S_i^* for S_i;
  | Obtain S_i' by removing those services from S_i dominated by S_i^*;
end
repeat
  | select one candidate service from each service group S_i';
  | form a composite service C_i;
  | if ∃C_i' ∈ R such that C_i' ≻ C_i with respect to p then
  |   | discard C_i;
  | else
  |   | insert C_i to R;
  |   | remove every service C_i' ∈ R such that C_i ≻ C_i' with respect to p;
  | end
until no more services can be traversed ;
Return R;
```

Based on the heuristic rule above, now we consider how to find such $S_i^{ti'}$s for service group S_i. For a service $S_i^{ti'}$ in S_i, its ability to prune and exclude other services from further consideration is determined by its own QoS value and the extent of the distribution of the QoS values of services in S_i. Suppose the value range on metric $k(k = 1, 2, 3, 4)$ is $[a_k, b_k]$ in S_i. Then the pruning ability of $S_i^{ti'}$, denoted as $PA(S_i^{ti'})$, is evaluated as $PA(S_i^{ti'}) = \prod_{l=1}^{4} (b_l - q_{ti'}^l)$. We select the service from S_i with the maximum pruning ability, termed S_i^* to prune non-qualifying services. After pruning non-qualifying services for each service group S_i, the straightforward strategy is applied to compute the query result $R(p, C)$. Let $S_i^{'}$ represent S_i from which some services have been pruned. The number of pruned services for S_i is $|S_i| - |S_i^{'}|$. Therefore, the total number of pruned composite services in C is $\prod_{l=1}^{k} (|S_l| - |S_l'|)$. Algorithm 3 shows the pseudo-code of the LPM algorithm.

It is important to point out that in the worst case the number of pruned services $|S_i| - |S_i^{'}|$ may equal to 0, which happens when none of candidate services are dominated by one other service in the same service group, i.e. all candidate services are skylines in the same group. In this special case, the LPM heuristics is invalid and we use the RC and RRC algorithms to prune local services.

3.2.3 More pruning method

Algorithm 4 The MPM algorithm for QoS query on composite services

```
input : A QoS query vector p
        k service groups S_1, S_2, ..., S_k
output: A query result set R containing satisfying composite services
1  R = φ;
2  foreach service service group S_i do
3  |  S_i' = RRC(p, S_i);
4  end
5  repeat
6  |  select one candidate service from each service group S_i';
7  |  form a composite service C_i;
8  |  if ∃C_i' ∈ R such that C_i' ≻ C_i with respect to p then
9  |  |  discard C_i;
10 |  else
11 |  |  insert C_i to R;
12 |  |  remove every service C_i' ∈ R such that C_i ≻ C_i' with respect to p;
13 |  end
14 until no more services can be traversed ;
15 Return R;
```

The *More pruning method* (MPM) also uses dominance checking to prune the traversal space, and the idea of pruning also applies to MPM. However, unlike LPM, which tries to find services in each service group that dominate other services, MPM will instead aim to locate those services that are not dominated by any other one. In this way, services can be further pruned from consideration. It is evident that the pruning result for a service group returned by MPM equals the query result of QoS query on the same service group. So, firstly we can use the proposed algorithms for QoS query on individual services to perform pruning, then traverse the left services

to find qualifying composite services. MPM calls the algorithm RRC to carry out the further pruning procedure. Slightly modifying the LPM algorithm, the MPM algorithm is shown in Algorithm 4.

4 Personalized service selection

Having shown QoS models and their corresponding computing algorithms for answering a user's QoS query on both individual services and composite services, we know that our strategy can solve the *empty result* problem, as shown in the introduction section, and can provide at least one result to satisfy the user's QoS requirement. But meanwhile, another problem appears: the number of services in the query result may exceed the user's requirement and sometime she or he does not need all services in the query result. For example, a strict user may only needs fully satisfying services; whereas a user running out of money prefers a relatively cheaper service with other QoS metrics being satisfied approximately. So, a natural question is how to select good or appropriate services from the returned query result to satisfy the user's personalized requirement. We categorize this question into two cases as follows.

4.1 Selecting fully satisfying services

For a strict user requiring fully satisfying services, a question to ask is whether these services are included in the returned query result or not. Actually, it is important to point that the query result specified by definition 3 or definition 7 is consistent with the user's quality requirements. This fact can be derived from the following lemma.

Lemma 1 *The query result of a QoS query vector p on a group of services S,* $R(p, S)$*, contains* ***at least one*** *fully satisfying service if such services exist.*

Proof Suppose F is a set of all the fully satisfying services. Obviously, there exists a service $a \in F$ such that a is not dominated by any other service in F. If $a \notin R(p, S)$, there must exist a non-fully satisfying service $b \in R(p, S)$, such that $b \succ a$. So, by definition 2 we conclude the value of $D(p, b)$ on each dimension is not larger than that of $D(p, a)$) and strictly smaller on at least one dimension, which is wrong because all the dimension values of $D(p, a)$ are strictly less than zero and at least one dimension value of $D(p, b)$ is larger than or equal to zero. □

Notice that not all fully satisfying services are contained in $R(p, S)$, as there may exist dominance relationship between them. By Lemma 1, we can simply select the fully satisfying services in $R(p, S)$ as the best answers for the strict user. Here we only discussed the case of individual services. For composite services, slightly modifying Lemma 1, we can obtain fully satisfying composite services similarly.

4.2 Top-k answers on out-of-range services

Since we try to obtain the query result using a relaxation method by computing the *quality difference*, rather than seeking services satisfying all the four conditions at

one time, our proposed models can still return services to answer a user's QoS query, even when no fully satisfying services are available. Reasonably, the user may only be interested in part of the returned out-of-range services. However, different users may have different preferences for one fixed metric, and even one user may have different preference for different metrics. So, an important problem is how to compute the k best out-of-range services.

In our models, the metrics are treated equally. In order to satisfy the user's personalized QoS requirements, we allow the user to specify a weight for each metric to adjust the *quality difference* when she issues QoS queries. Specifically, the quality difference of a service s with respect to a query vector p is modified as $D(p, q(s)) = (w_1 \times \delta_1, w_2 \times \delta_2, w_3 \times \delta_3, w_4 \times \delta_4)$, where $w_i (i = 1, 2, 3, 4)$ are weights specifying the importance of the quality difference on each metric. Furthermore, we define the user's preference score for s as $w_1 \times \delta_1 + w_2 \times \delta_2 + w_3 \times \delta_3 + w_4 \times \delta_4$. Thus, after the out-of-range service set R is available, we can select k services in R with the k smallest preference scores to satisfy the user's top-k requirement.

The step of finding the top-k answers can also be combined into the algorithms for computing $R(p, S)$ or $R(p, C)$. To achieve this, we use a heap to store the k best services obtained so far. When a new candidate service is available, we compute its preference score and update the heap. For the limit of space, the detailed procedure is omitted here.

5 Experiments

In this section, experiments are conducted to evaluate the performance of the proposed algorithms. The experiments were implemented on a P4 Windows XP machine with a 2 GHz Pentium IV and 512M main memory. Our experiments are divided into two groups. Group 1 is for evaluating the performance of naïve , RC and RRC algorithms for QoS query on individual services. Group 2 is for evaluating the performance of algorithms for QoS query on composite services, including straightforward method, LPM and MPM. We use the simulation approach in [27] to study the performance of these algorithms. The comparisons of these algorithms are done by running time (Figure 3).

We first evaluate the efficiency of algorithms running on individual services. As we have seen, the two algorithms naïve and RC do not require index structure to carry out queries but the RRC algorithm firstly needs to construct an R-tree to index all the QoS vectors of a service group. Therefore, we need to investigate the performance of constructing a R-tree for RRC algorithm and then compare the query performance of these three algorithms. The result is shown in Figure 4.

For simulation, we generate a group of candidate services, the number of which ranging from 1,000 to 5,000. Four quality values for each service are randomly generated with a uniform distribution between [0, 1]. Figure 3a shows the result of the three proposed methods as the number of services increases. The test result shows RC is much better than the naïve method. The reason is that in the naïve strategy every pair of services is compared without any pruning. RRC is faster than RC, especially when the services are large. This is because RRC uses R-tree to index all service quality vectors, and the number of comparisons is reduced largely with its minimum bounding rectangles (MBRs). It is worth noting that when taking into

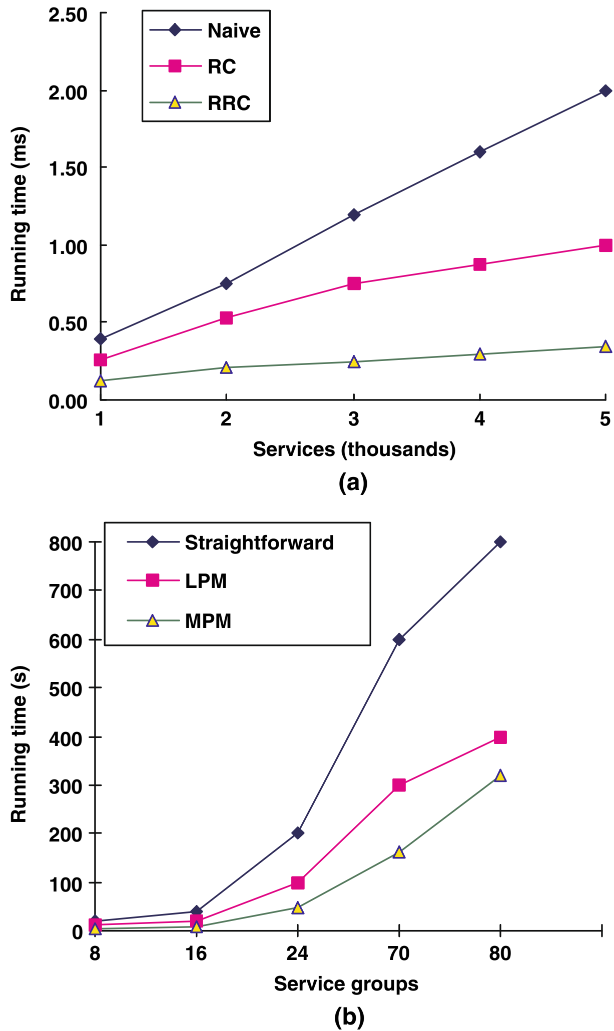


Figure 3 Running time comparisons on individual and composite services.

account the time spent in constructing the R-tree, RRC is slower than RC as building a R-tree is a time consuming process. However, as once the service groups are available they are relatively static and there are not much update or change involved, the indexing process only needs to be done once. Considering service selections are query operations in most cases, the RRC will achieve good performance once its index has been available.

Then, the algorithms for QoS query on composite services are evaluated. We randomly construct 80 groups of services, each group having 100 candidate services. The results of straightforward, LPM and MPM algorithms are shown in Figure 3b. As can be seen, the straightforward method is the most expensive because it has to traverse all services in each service group, which leads to the exponential increase of running time. LPM performs much better, indicating that by pruning the number of services in each group, the efficiency of finding qualifying composite services

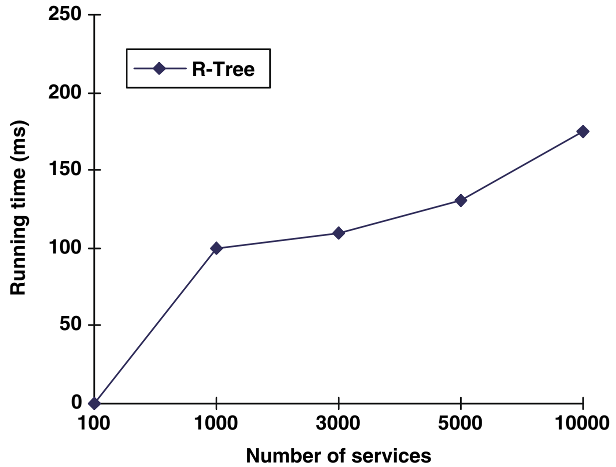


Figure 4 Running time of building the R-tree for RRC.

improves significantly. It is also can be seen that the performance of LPM is low compared with MPM. This demonstrates pruning services of each service group as many as possible is very beneficial. Although MPM spends time to compute QoS query on each service group first, its performance is still good due to fact that the number of services in each service group is a key factor in the cost.

Finally, we evaluate the performance of the method proposed to obtain the top-k answers from the services contained in $R(p, S)$. As the step of finding the top-k answers can also be combined into the two algorithms, its performance is determined by the number of returned services, as shown in Figure 5. It can be seen that as the number of returned services in $R(p, S)$ increases, the running time for calculating the top-k personalized services goes up. Please note the experiment of top-k algorithm is carried out based on the returned query result and its cost does not include the time spent on calculating $R(p, S)$.

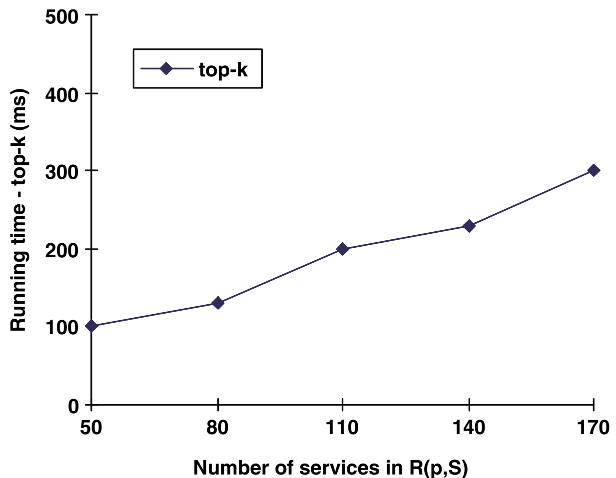


Figure 5 Running time for different size of returned services.

6 Related works

QoS-based web service selection is an active research area and has attracted many researchers [3, 9, 18, 19]. A lot of methods are proposed in previous work, such as workflow model, global planning strategy using linear programming [10], graph/tree approach [2], QoS based selection of semantic web services, etc. In this section, we briefly discuss the relationships between our work and existing methods.

Industrial standard specifications have been proposed to provide infrastructure for web services composition. Among them BPEL4WS (Business Process Execution Language for Web services) is the most widely used language for process-based service composition [7, 29]. It describes the execution procedure and abstract process of workflow. Other specifications include E-Flow, BPML, etc. However, these specifications only provide some approaches to carry out local services selection in dynamic environment, and QoS optimization is not supported. Our work is based on these proposals and aims to provide a QoS-based and personalized services selection model for the underlying workflow.

Besides industrial standards, many prototypes have been developed to assist in services composition. In particular, SWORD uses a rule-based engine to realize a composition by existing web services; SELF-SERVE proposes a declarative language to carry out service composition based on state-chart. But these projects only focus on planning or analyzing workflow process, and neither QoS criteria nor QoS optimization issues are addressed.

QoS-based service selection could be considered a special case of the more general problem of global optimization. [8] presents a QoS model, addressing time, cost, and reliability dimensions. The model computes the quality of service for workflows automatically based on QoS attributes of an atomic task. In [28], the QoS of web services is computed using a muti-dimensional model, and the global QoS optimization is solved by linear programming techniques. In [26], authors present a QoS broker to maximize the user-defined utility value. In the broker, the optimized services selection is modeled as the Multiple Choice Knapsack Problem and the shorted path problem in graph theory. In [23], authors use an AND/OR tree structure to model the service composition problem. The procedure of service composition is implemented through tree traversal, and a heuristic-based search method is proposed to retrieve composite services with top-k QoS values. Wang et al. [21] uses a qualitative graphical representation of preference, CP-nets, to deal with services selection in terms of user preferences. Wang et al. [22] proposed several heuristic algorithms to decompose the general service composition request graph into service composition request subgraphs with optimal structures. Authors in [9] describe a new user centric service-oriented modeling approach which is featured by integrating fuzzy technique to an Ideal Solution (TOPSIS) and Service Component Architecture (SCA) to facilitate web service selection and composition and to effectively satisfy a group of service consumers requirements. However, these approaches can not handle the cases where there is no fully satisfying service available, and the personalization issue is not discussed either.

Several works are related to QoS-based selection of semantic web services [16]. Specifically, DAML-S and ebXML provide well defined, computer-interpretable semantics for web services. Wang et al. [24] describes a QoS model using the Web Service Modeling Ontology. Also, the idea of QoS model extension have been

presented by some researchers, such as [6, 13], etc. In [6], the extended QoS model includes generic and domain or business specific criteria and allows users to express their preferences. Jurca et al. [13] models web service configurations, associated prices and preferences using utility function policy, and the optimal service selection combines declarative logic-based matching rules with linear programming optimization methods. However, these works do not solve the empty result problem either. Furthermore, although users' interest is concerned, it is only based on one preference and thus infeasible for practical purpose. To the contrary, our model takes into account all QoS criteria factors and a relaxation strategy is applied to describe users' comprehensive preference.

Another area related to this paper is skyline query [5, 14, 17, 25], for example, [5] proposed the skyline operator; [17] developed an optimal and progressive algorithm for skyline queries; [14] introduced a skyline framework for defining the semantics of selection and join queries on relational database. Inspired by these works, we use a relaxation-based approach to perform QoS-based web service selection.

7 Conclusions

In this paper we studied the problem of service selection with QoS constraints. A novel QoS model was proposed to perform flexible service selection. Based on the presented model, we developed various algorithms for making service selection on individual and composite services, respectively. We also introduced a top-k ranking strategy to reflect a user's personalized requirement. The performance of the algorithms has been evaluated, showing the proposed QoS model is an efficient and practical strategy to satisfy users' QoS requirements. As part of on-going work, we are interested in improving performance of the QoS query algorithms, as well as investigating more complicated workflow patterns. We also plan to integrate exception handling mechanism into our model during personalized service selection.

References

1. Benatallah, B., Casati, F.: Guest editorial. Distrib. Parallel Dat. **12**(2/3), 115–116 (2002)
2. Bertino, E., Squicciarini, A.C., Paloscia, I., Martino, L.: Ws-ac: a fine grained access control system for web services. World Wide Web **9**(2), 143–171 (2006)
3. Bianchini, D., Antonellis, V.D., Melchiori, M.: Flexible semantic-based service matchmaking and discovery. World Wide Web **11**(2), 227–251 (2008)
4. Booth, D., Haas, H., McCab, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web services architecture. http://www.w3.org/tr/ws-arch/ (2004)
5. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
6. Canfora, G., Penta, M.D., Esposito, R., Perfetto, F., Villani, M.L.: Service composition (re)binding driven by application-specific qos. In: ICSOC, pp. 141–152 (2006)
7. Cao, J., Zhao, H., Li, M., Wang, J.: A dynamically self-configurable service process engine. World Wide Web **13**(4), 475–495 (2010)
8. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. J. Web Semant. **1**(3), 281–308 (2004)
9. Cheng, D.Y., Chao, K.M., Lo, C.C., Tsai, C.F.: A user centric service-oriented modeling approach. World Wide Web **14**(4), 431–459 (2011)

10. Cibrán, M.A., Verheecke, B., Vanderperren, W., Suvée, D., Jonckers, V.: Aspect-oriented programming for dynamic web service selection, integration and management. In: World Wide Web, pp. 211–242 (2007)
11. Huang, Z., Jensen, C.S., Lu, H., Ooi, B.C.: Skyline queries against mobile lightweight devices in manets. In: ICDE, p. 66 (2006)
12. Jaeger, M.C., Rojec-Goldmann, G., Mühl, G.: Qos aggregation for web service composition using workflow patterns. In: EDOC, pp. 149–159 (2004)
13. Jurca, R., Faltings, B., Binder, W.: Reliable qos monitoring based on client feedback. In: WWW, pp. 1003–1012 (2007)
14. Koudas, N., Li, C., Tung, A.K.H., Vernica, R.: Relaxing join and selection queries. In: VLDB, pp. 199–210 (2006)
15. Liu, Y., Ngu, A.H.H., Zeng, L.: Qos computation and policing in dynamic web service selection. In: WWW (Alternate Track Papers & Posters), pp. 66–73 (2004)
16. Martin, D.L., Burstein, M.H., McDermott, D.V., McIlraith, S.A., Paolucci, M., Sycara, K.P., McGuinness, D.L., Sirin, E., Srinivasan, N.: Bringing semantics to web services with owl-s. In: World Wide Web, pp. 243–277 (2007)
17. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: SIGMOD Conference, pp. 467–478 (2003)
18. Rasch, K., Li, F., Sehic, S., Ayani, R., Dustdar, S.: Context-driven personalized service discovery in pervasive environments. World Wide Web **14**(4), 295–319 (2011)
19. Schewe, K.D., Thalheim, B., Wang, Q.: Customising web information systems according to user preferences. World Wide Web **12**(1), 27–50 (2009)
20. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Dat. **14**(1), 5–51 (2003)
21. Wang, H., Xu, J., Li, P.: Incomplete preference-driven web service selection. In: IEEE SCC (1), pp. 75–82 (2008)
22. Wang, J., Wang, J., Chen, B., Gu, N.: Minimum cost service composition in service overlay networks. World Wide Web **14**(1), 75–103 (2011)
23. Wang, X., Huang, S., Zhou, A.: Qos-aware composite services retrieval. J. Comput. Sci. Technol. **21**(4), 547–558 (2006)
24. Wang, X., Vitvar, T., Kerrigan, M., Toma, I.: A qos-aware selection model for semantic web services. In: ICSOC, pp. 390–401 (2006)
25. Yang, J., Fung, G.P.C., Lu, W., Zhou, X., Chen, H., Du, X.: Finding superior skyline points for multidimensional recommendation applications. World Wide Web **15**(1), 33–60 (2012)
26. Yu, T., Lin, K.J.: Service selection algorithms for web services with end-to-end qos constraints. Inf. Syst. E-Business Management **3**(2), 103–126 (2005)
27. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. TWEB **1**(1) (2007)
28. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW, pp. 411–421 (2003)
29. Zhao, X., Liu, C., Sadiq, W., Kowalkiewicz, M., Yongchareon, S.: Implementing process views in the web service environment. World Wide Web **14**(1), 27–52 (2011)