7-2009

A Relational Approach for Efficient Service Selection

Qi Yu
Rochester Institute of Technology

Manjeet Rege
Rochester Institute of Technology

Recommended Citation

Q. Yu and M. Rege, "A Relational Approach for Effcient Service Selection," 2009 IEEE International Conference on Web Services, Los Angeles, CA, 2009, pp. 719-726. doi: 10.1109/ICWS.2009.33

A Relational Approach for Efficient Service Selection

Qi Yu
Rochester Institute of Technology
qi.yu@rit.edu

Manjeet Rege
Rochester Institute of Technology
mr@cs.rit.edu

Abstract

Web services are gaining momentum as a major vehicle to deliver business functionalities on the Web. More and more business organizations have begun to use Web services to facilitate user interactions and the collaboration among themselves. This essentially forms a large service space, which still keeps growing. Meanwhile, there may be functionality overlaps among different service providers. The concept of Quality of Web Serivce (QoWS) *is emerging as a key feature in distinguishing between competing service providers. We present in this paper a systematic approach for efficient service selection by using QoWS as the major criterion. In particular, we adopt a relational approach that enables to store QoWS information in a relational DBMS and leverage standard relational operators for efficient service selection. We perform a preliminary set of experiments to evaluate the proposed service selection algorithms.*

1 Introduction

As the Web is moving from a *data Web* to a *service Web*, it is expected that tomorrow's Web will be the repository of a large number of Web services provided by third party providers [13]. In that context, the ability to *efficiently* select and access Web services is poised to become of prime importance. In the simplest scenario, accessing Web services would consist of invoking their operations by sending and receiving messages. However, for complex applications (e.g., a travel package), there would be a need for an *integrated* and *efficient* approach to *select* and *deliver* Web services' functionalities.

Let's consider a trip planning example. The services that may be useful when planning a trip would include *TripPlanner*, *Map*, and *Weather*, etc. The *TripPlanner* service offers some basic information, such as airlines, hotels, and local attractions. The *Map* service offers users the geographic information, local attractions, etc. The *Weather* service would also be relevant for getting the weather condition during the trip. Each service may consist of multiple operations and there may be dependency relationship between these operations. For example, the map service may have two operations *GeoCode* and *GetMap*. *GetMap* will rely on *GeoCode* to generate its result.

When accessing each of these services, a typical user usually prefers to using a service provider with their desired quality (e.g., fee, response time, and reputation, etc). However, this usually requires a series of trial-run processes and would be very painstaking if the number of competing providers is large. Things will become more complicated when the user needs to consider these services together as a whole package. The possible combinations of providers for the service package will far exceed the range for a manual selection by the users. Thus, users could easily miss their desired service package.

From the above example, we can see that as Web services with *similar* functionality are expected to be provided by competing providers, a major challenge is devising service selection mechanisms for choosing the "best" Web services or their compositions. The concept of *QoWS* is considered as a key feature in distinguishing between competing Web services [9, 11]. *QoWS* encompasses different quality parameters that characterize the behavior of a Web service in delivering its functionalities. Examples of parameters include availability, latency, and fees.

Several service selection techniques have been investigated [6, 15, 12, 14, 5, 10]. Typically, the QoWS-based service selection relies on computing a score function $\mathcal{F}(\vec{q}, \vec{w})$, where $\vec{q}$ is a set of quality of service parameters and $\vec{w}$ is a set of weights assigned for each parameter in $\vec{q}$. The score function $\mathcal{F}$ assigns a scalar value to each service provider and the provider gaining the highest value will be selected and returned to the user. Existing service selection approaches usually employ a two-phase process. In the first phase, a service composition plan is developed by using a set of abstract services. The second phase will instantiate this plan by using the actual service providers. The QoWS information from these providers will be used to compute the score function and the plan with the highest score will be selected. These approaches assume that the service providers fully conform to the abstract service. That

is, they offer the same operations defined by the abstract service and follow the dependency constraints. However, a large number of service providers may partially match the abstract services. For example, a service provider may offer a subset of operations defined by a abstract service. This provider could still be useful when this subset of operations can fulfill a user's functionality requirement.

We present in this paper a service selection approach that can deal with service providers that partially conform to the abstract services. This essentially enable to deploy the service selection approach to more practical scenarios. It guarantees that a service provider can be selected if it provides the user required functionality even though it does not necessarily match the specification of the abstract service. In particularly, the service selection approach is built upon the service query optimization framework presented in [12]. A service model is provided by this framework that define a service schema and a service relation. The service schema plays a similar role as abstract services. The service relations are used to store QoWS information of service providers. We devise a relational approach to efficiently select the best service providers based on the service model. More specifically, we present two ways to decompose a service relation: Q-Decompose and O-Decomose. The results will be a set of Q-relations or O-relations. Service selection can be easily achieved by just using standard relational operators. The O-relations offer a more compact representation of the original service relation but require a set of expensive join operations to get the desired service providers. We present to use a bitmap structure to enable efficient access to the O-relations.

The remainder of this paper is organized as follows. The proposed service selection mechanism is developed based upon a formal service model, which we will briefly describe in Section 2. We then present the system architecture for service selection. We present the service selection algorithms in Section 3. We experimentally evaluate the proposed algorithms in Section 4. We overview the related work in Section 5 and provide some concluding remarks in Section 6.

2 The Web Service Framework

In this section, we briefly introduce the service query optimization framework that was first presented in [12]. The framework defines a formal service model, based on which we develop the service selection mechanisms. We also describe the system architecture for service selection.

2.1 The Service Model

We describe the service model in this section, which formally defines two key concepts: *service schema* and *service relation*.

DEFINITION 1 **(Service Schema [12]).** A service schema $\mathcal{S}$ is defined as a tuple $(SG_1, ..., SG_n, \mathcal{D})$, where each SG_i is a DAG, called *service graph*. In $SG_i = (V_i, E_i, \epsilon_i)$, the vertex set V_i represents the set of service operations in the service graph, the edge set E_i represents the dependency constraints between service operations, and ϵ_i is the root of the service graph representing the entry point, through which all other operations in the service graph can be accessed. $\mathcal{D}$ represents the set of dependencies between two non-root operations from different service graphs. ■

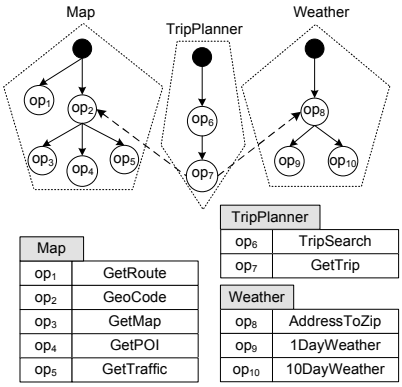

Figure 1. The Service Schema for Trip Planning

Figure 1 shows the service schema for our trip planning example. The service schema contains three service graphs, representing the *Map*, *TripPlanner*, and *Weather* services. For example, in the *Map* service, there are a set of service operations, such as *GeoCode*, and *GetMap*, etc. These operations collectively represent the functionality of the *Map* service. The dependencies between service operations are captured by the edges in the service graph. For example, (op_2, op_3) means that the execution of *GetMap* depends on the result of *GeoCode*. Service operations from different Web services could have an inter-service dependency. For example, there is a dependency between *GetTrip* and *GeoCode*. It is denoted by (op_7, op_2).

Having the above service schema, users only need to specify the operation(s) they want to access (i.e., in a declarative way) in a service query. Since an operation can only be invoked after the invocation of all its dependent operations, a key concept called *operation graph* is introduced to capture these operations and the dependencies between them.

DEFINITION 2 **(Operation Graph [12]).** For a service graph $SG = (V, E, \epsilon)$, an operation graph $G(op)$ is the *union* of all the paths in SG that lead to operation op. $G(op)$ is a *subgraph* of the service graph SG. Figure 2 shows

an operation graph $G(d)$, which is formed from SG by the union of two paths, P_1 and P_2, that both lead to the service operation d. ∎

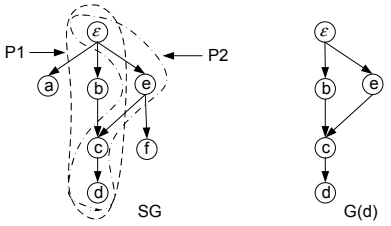

Figure 2. An Example of An Operation Graph

Given a service operation and a service graph, the operation graph can be directly obtained through standard graph algorithms. There are two key extensions on the operation graph that allows users to access multiple service operations in an integrated manner. *Operation set graph*, which is defined as $G(\vec{op}) = \cup_{i=1}^{k} G(op_i)$ (where $\vec{op} = \{op_i | 1 \leq i \leq k\}$), is used to access multiple operations from the same service graph. When a service query accesses service operations from different service graphs, we define the *composition of operation graphs*, $G' = G_i \circ G_j$. G' is formed by coalescing the root of G_i and G_j. The inter-graph edges become part of the edge set V' in the newly formed service graph G'. The newly generated root needs to store the entry information (e.g., URI) for accessing service operations from original service graphs.

The service relation is used to capture the quality of different service providers. It defines a set of service instances that conform to the service schema. The service instances offer the operations and follow the dependency constraints defined in the service graphs. However, since the service instances are provided by different service providers, they may have different quality properties.

DEFINITION 3 **(Service Relation [12]).** A service relation SR with a service graph $SG = (V, E, \epsilon)$ is defined as a set of service instances $\mathcal{I} = \{(sid, op_1, ..., op_n)\}$, where sid is the unique service id; op is a service operation and defined as a pair $op = (opid, \mathcal{Q}(op))$, where $opid$ is the operation id and $\mathcal{Q}$ is a set of QoWS parameters of op. Table 1 gives the definition of each QoWS parameter. ∎

2.2 Service Execution Plans

Given an operation graph, we can perform a *topological sort* on the operation graph, which will order the operations based on their dependencies. This operation sequence is referred to as a *generic service plan*. SEPs can be generated by instantiating the generic service plan with the operations from the service instances in the service relation. The quality parameters of a SEP can be computed by aggregating those of its member service operations. These aggregation functions are widely used in dealing with various service optimization problems [14, 15, 12]. Table 2 shows these aggregation functions.

Table 2. QoWS for a SEP

QoWS parameter	Aggregation function
lat(SEP)	$\sum_{i=1}^{n} lat(op_i)$
rel(SEP)	$\sum_{i=1}^{n} \log(rel(op_i))$
avail(SEP)	$\sum_{i=1}^{n} \log(avail(op_i))$
fee(SEP)	$\sum_{i=1}^{n} fee(op_i)$
rep(SEP)	$\frac{1}{n} \sum_{i=1}^{n} rep(op_i)$

The following score function F can be used to select the best SEPs.

$$F = \left(\sum_{Q_i \in neg} W_i \frac{Q_i^{max} - Q_i}{Q_i^{max} - Q_i^{min}} + \sum_{Q_i \in pos} W_i \frac{Q_i - Q_i^{min}}{Q_i^{max} - Q_i^{min}} \right)$$

neg and pos represent negative and positive *QoWS* respectively. In negative (resp. positive) parameters, the higher (resp. lower) the value, the worse is the quality. W_i are weights assigned by users to each parameter. Q_i is the value of the i^{th} *QoWS* of the service execution plan obtained through the aggregate functions from Table 2. Q_i^{max} is the maximum value for the i^{th} *QoWS* parameter for all potential service execution plans and Q_i^{min} is the minimum. These two values can be computed by considering the operations from service instances with the highest and lowest values for the i^{th} *QoWS*.

2.3 System Architecture

The system architecture is illustrated in Figure 3. The service schema is derived from an application domain based on the major operations offered by these services and the dependencies between these operations. The QoWS monitor is in charge of collecting quality information from service providers. Some QoWS monitoring mechanisms that have been investigated recently [4, 1] can be used for this purpose. The service relations conform to the service schema. Each service graph will have a corresponding service relation. The service relation will store the QoWS information of all operations that appear as a node in the service graph. The plan generator is used to generate the generic service plan. The plan optimizer will select the best SEP based on the QoWS information stored in the service relations. Finally, the plan processor will interact with the selected providers to perform the user's task.

Table 1. QoWS Parameters

Parameter	Definition	Abbr.	Index
Latency	Time$_{process}$(*op*) + Time$_{results}$(*op*) where Time$_{process}$ is the time to process *op* and Time$_{results}$ is the time to transmit/receive the results	*lat*	1
Reliability	N$_{success}$(*op*)/N$_{invoked}$(*op*) where N$_{success}$ is the number of times that *op* has been successfully executed and N$_{invoked}$ is the total number of invocations	*rel*	2
Availability	UpTime(*op*)/TotalTime(*op*) where UpTime is the time *op* was accessible during the total measurement time TotalTime	*avail*	3
Fee	Dollar amount to execute the operation	*fee*	4
Reputation	$\sum_{u=1}^{n} Ranking_u(op)/n$, $1 \leq$ Reputation ≤ 10 where $Ranking_u$ is the ranking by user *u* and n is the number of the times *op* has been ranked	*rep*	5

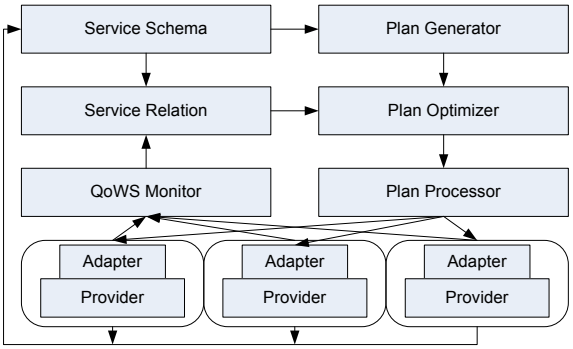

Figure 3. System Architecture

3 Dealing with Partially Matching Providers

The service model offers a high level abstraction of various service providers in an application domain. More specifically, the service schema captures the dependencies between different service operations. These dependencies can be usually derived from the business logics commonly accepted in the given domain. For example, in the travel industry, if you reserve a package that includes both airline booking and car rental, the pickup time and location of the rental car is automatically determined by the air ticket. Such kind of business logics will be used in developing a service model for the travel related Web services.

In essence, the service schema provides a layer of abstraction on top of the actual service providers. The technical details like how many service providers are out there and how to access these providers are all transparent from the service users. Since the users only interact with the service model, it is necessary to include all the operations that are commonly used in the domain of interest. In another word, if some of these operations are not part of the service schema, the service users will not have access to them. However, this will introduce another issue because many service providers may just implement a subset of operations defined by the service schema. In fact, this could be quite common. For example, a map service provider A may just offer the map and routing operations; another service provider B may offer map and POI operations. In this regard, these providers only partially match the service schema. For users that are only interested in routing or POI, these services still have their merits. However, if a strict matching to the schema is required, many service providers (like A and B) are not able be included in the system.

An intuitive solution for the partially matching providers is to have multiple specific service graphs for each service instead of just having one general service graph. For example, we could have a service graph Map$_A$ that only includes op_1 in Figure 1 and another service graph Map$_B$ that only includes op_2 and op_3. Thus, provider A matches Map$_A$ and provider B matches Map$_B$. A major issue with this intuitive solution is that the number of service graphs will increase exponentially if all possible situations are to be considered. More specifically, 2^t service graphs need to be generated for a graph with t leaf nodes. Another issue is that since each service graph has a corresponding service relation, a corresponding number of service relations need to be generated as well and the QoWS information for many providers will be duplicated across these service relations.

In this section, we present a disciplined service selection approach to deal with these partially matching providers. In this approach, we stick with one general service graph for each service. Before describing the proposed approach, we first introduce the concept of service relation decomposition.

3.1 Service Relation Decomposition

A service relation is not in the First Normal Form (1NF) because a service operation is a composite attribute, which consists of an operation id (opid) and a set of QoWS values [12]. The decomposition process helps normalize a service relation into a set of relations that satisfy 1NF so that

we can use a relational approach to deal with the partially matching providers. A service relation can be decomposed along two different dimensions: QoWS and service operation. We refer to these decompositions as *Q-Decompose* and *O-Decompose*, respectively.

3.1.1 Q-Decompose

Q-decompose will transform a service relation into a set of QoWS relations, referred to as *Q-relations*. For example, the map service relation will turn into five QoWS relations, Map_Latency, Map_Availability, Map_Reliability, Map_Fee, and Map_Reputation. Figure 4 shows the schema for each QoWS relation, where sid is the unique id for a service provider and each op column stores the QoWS information for that operation. For example, column op_1 in Map_Latency stores the latency for operation op_1.

Map_Latency

sid	op1	op2	op3	op4	op5

Map_Availability

sid	op1	op2	op3	op4	op5

Map_Reliability

sid	op1	op2	op3	op4	op5

Map_Fee

sid	op1	op2	op3	op4	op5

Map_Reputation

sid	op1	op2	op3	op4	op5

Figure 4. Q-Decompose of Map Service Relation

Since all these Q-relations are in 1NF, we can store them in a relational database. In this regard, if a service provider does not offer a certain operation, say op_i, defined in the service graph, we can simply put a "null" value in the op_i column. Figure 5 shows how to represent partially matching providers in Q-relations. For example, since service provider A only offers the routing operation (i.e., op_1), operations op_2 to op_5 all take a "null" value. A potential issue with this approach is how to differentiate with a missing QoWS value. For example, a service provider D offers all the operations defined in the service graph. However, the latency for op_5 is missing. In this case, provider D has a "null" value for op_5. The difference between D and A (or B) is that if a user wants to access op_5, D is still a potential provider to be considered [1]. However, A and B should not be selected at all because they do not offer op_5. The objective is to make sure that the providers are not selected when the users want to access the operations that are not offered by them. To achieve this effect, we can use $+\infty$ for negative QoWS parameters and $-\infty$ for positive QoWS parameters. The SEPs that have such operations will always end up with a negative value for their objective function and thus can never be selected.

[1]How to deal with missing QoWS information in service selection is out of scope of this paper.

Map_Latency

sid	op1	op2	op3	op4	op5
A	5	null	null	null	null
B	6	1	null	null	null
C	7	3	1	2	1
D	5	1	3	2	null

Figure 5. Representing Partially Matching Providers in Q-Decompose

3.1.2 O-Decomose

O-decomose will transform a service relation into a set of operation relations, referred to as *O-realtions*. For example, the map service relation will turn into five operation relations through O-decompose: Map_op_1, Map_op_2, Map_op_3, Map_op_4, and Map_op_5. Each of these relations will contain six columns, including sid and the five QoWS parameters. If a Map service provider does not offer operation op_i, we can simply exclude it from the O-relation Map_op_i. Figure 6 shows the Map_op_1 O-relation. As can be seen, since provider B does not offer op_1, it is not included in this operation relation.

Map_op1

sid	Latency	Availability	Reliability	Fee	Reputation
A	5	0.9	0.8	2	4
C	7	0.8	0.7	3	3
D	5	0.9	0.7	2	4

Figure 6. Representing Partially Matching Providers in O-Decompose

3.2 Service Selection

In this section, we present the service selection algorithms based on the decomposed service relations.

3.2.1 Service Selection with Q-Decompose

The first algorithm performs on the Q-relations generated by Q-Decompose. Thus, the input of this algorithm will be a set of Q-relations and a generic service plan GSP. Assume that the service operations are from m services and there are k_i operations from service S_i: $op_{i,1}, ..., op_{i,ki}$. The algorithm consists of three phases (as shown in Algorithm 1). In phase I, the user selected operations are retrieved from each

Q-relation of S_i. $Q_{i,j}$ is the j^{th} Q-relation, where $j \in [1,5]$ is the index of QoWS parameters defined in Table 1. The results are stored in a set of relations $\{R_{i,j}|j \in [1,5]\}$. Then, in Phase II, the operations from different services are combined to form the candidate SEPs. Specifically, we perform a Cartesian product among Q-relations $R_{1,j}$ to $R_{m,j}$ and store the result in R_j. For example, at the end of Phase II, Q-relation R_1 will contain the latency of all operations in the SEPs. In Phase III, the QoWS will be aggregated based on the aggregation functions defined in Table 2 and the result will be stored in a set of aggregation relations $\{Agg_j|j \in [1,5]\}$. For example, Agg_1 and Agg_5 will store the latency and reputation of all SEPs, respectively. Finally, we compute the objective function F based on the QoWS of SEPs and select the SEP with the highest F value. As can be seen, for service providers that do not offer certain service operations, the QoWS of these operations will be set to $+\infty$ (for negative QoWS parameters) or $-\infty$ (for positive QoWS parameters). Thus, the SEP that contains those operations will have a negative F value and will automatically be excluded.

Algorithm 1 Service Selection with Q-Decompose

Input: m services and the k_i operations from each service S_i: $op_{i,1}, ..., op_{i,ki}$

1: **Phase I: Select operations from each** S_i
2: **for all** $i \in [1,m]$ **do**
3: **for all** Q-relation $Q_{i.j}$ **do**
4: $R_{i,j} = \pi_{op_{i,1},...,op_{i,ki}}(Q_{i,j})$ // projection
5: **end for**
6: **end for**
7: **Phase II: Combine different services**
8: **for all** $j \in [1,5]$ **do**
9: $R_j = R_{1,j}$
10: **for all** $i \in [2,m]$ **do**
11: $R_j = R_j \times R_{i,j}$ // Cartesian product
12: **end for**
13: **end for**
14: **Phase III: Aggregate the results**
15: **for all** $j \in [1,5]$ **do**
16: $Agg_j = \sum_{i=1}^{n_j} R_j(i)$ // n_j is the number of columns in R_j
17: **end for**
18: Compute F by using the aggregated QoWS values stored in Agg_j

3.2.2 Service Selection with O-Decompose

O-Decomose results in a set of O-relations. O-relations offer a more compact representation of the service relation because they no longer need to store $+\infty$ or $-\infty$ for partially matching providers. If a service provider does not offer op_i, it simple does not appear in the corresponding O-relation. However, service selection with O-relations will be a little bit more complicated. In order to find a provider for service S_i that offers k_i operations $op_{i,1}, ..., op_{i,ki}$, we need to search k_i O-relations. Only the provider with a sid that appears in all these O-relations will be retrieved. To achieve this, in Phase I, we perform an equal join across all the O-relations $O_{i,1}, ..., O_{i,k_i}$ that correspond to $op_{i,1}, ..., op_{i,ki}$ (as shown by Lines 2-5 in Algorithm 2). We then perform a Cartesian product to combine operations from different services. At the end of Phase II, we will have a single relation R, in which each tuple contains the QoWS information of all the operations in a candidate SEP. Then in Phase III, we can first aggregate the QoWS from all the operations to get the QoWS of the SEPs. We then compute the F values and select the best SEP.

Multi-table join is a very expensive process. The dynamic programming (DP) strategy is typically used by most DBMS to find optimal join order. This will greatly reduce the search space. However, in order to join k_i O-relations, DP still has a complexity of $O(2^{k_i})$. We propose a bitmap structure to improve the performance of service selection by using O-relations. Figure 7 shows the bitmap for the map service relation. Specifically, (p, op_i) will be 1 if a provider p offers operation op_i. It will be 0 if otherwise. To get the providers that offer $op_{i,1}, ..., op_{i,ki}$, we can simply perform a bitwise AND operation based on the bitmap of service S_i: $op_{i,1}\&...\&op_{i,ki}$. For example, to get the providers that offer operations op_1, op_2, and op_3 in the map service, we perform bitwise AND among op_1, op_2, and op_3: $1111\&0111\&0011$. The result is 0011, which means that providers C and D offer operations op_1, op_2, and op_3. Performing bitwise AND among $op_{i,1}, ..., op_{i,ki}$ has a complexity of $O(k_i)$. Having these providers, we just need to perform a set of selections (instead of expensive joins) on the operation relations to get their QoWS.

Algorithm 2 Service Selection with O-Decompose

Input: m services and the k_i operations from each service S_i: $op_{i,1}, ..., op_{i,ki}$

1: **Phase I: Select providers of** S_i **that offers** $op_{i,1}, ..., op_{i,ki}$
2: $O_i = O_{i,1}$
3: **for all** $j \in [2,k_i]$ **do**
4: $O_i = O_i \bowtie_{O_i.sid=O_{i,j}.sid} O_{i,j}$ // equal join
5: **end for**
6: **Phase II: Combine different services**
7: $R = O_1$
8: **for all** $i \in [2,m]$ **do**
9: $R = R \times O_i$
10: **end for**
11: **Phase III: Aggregate the results**
12: **for all** $j \in [1,5]$ **do**
13: Aggregate the j^{th} QoWS from different operations and put them into Agg_j
14: **end for**
15: Compute F by using the aggregated QoWS values stored in Agg_j

Bitmap

sid	op1	op2	op3	op4	op5
A	1	0	0	0	0
B	1	1	0	0	0
C	1	1	1	1	1
D	1	1	1	1	1

Figure 7. Bitmap for Map Service Relation

4 Experimental Study

We conducted a set of preliminary experiments to assess the performance of the proposed optimization approaches. We use the trip planning example as our testing environment to setup the experiment parameters. We run our experiments on a cluster of Sun Enterprise Ultra 10 workstations under Solaris operating system.

We create a service schema containing three service graphs, *TripPlanner*, *Map*, and *Weather*. We implement a service relation for each service graph. We use all the five QoWS parameters to evaluate service operations: latency, reliability, availability, fee, and reputation. The values of these parameters are generated within a range based on uniform distribution. The number of service providers in each service relation varies from 10 to 50. Each provider will a probability of p to offer a service operation. We will vary p from 0.5 to 0.9. More specifically, when creating each operation for a provider, we generate a random number r within the range of 0 to 1. $r \geq p$ implies that the provider offers this operation and we will go ahead to generate the QoWS information for this operation. $r < p$ implies that the provider does not offer this operation.

We decompose the service relations by using both Q-Decomopse and O-Decompose and store the resultant Q-relations and O-relations in a MySQL DBMS. We create a bitmap structure to accelerate the access to the O-relations. Assume that a user wants to access the following services and operations in our trip planning example: *Map (Geocode, GetMap)*, *TripPlanner (SearchTrip, GetTrip)*, and *Weather (AddressToZip, 10DayWeather)*.

Figure 8 shows how the CPU time varies with the number of service providers within each service relation. We set the probability that a provider offers an operation as 0.7, i.e., $p = 0.7$. As can be seen, the O-Decompose strategy offers a much better performance than the Q-Decompose strategy. The performance difference is due to the following reason. In Q-relations, we use some special place holders ($+\infty$ or $-\infty$) to avoid a provider that does not offer user required operations to be selected. The result is that a SEP that contains such providers will have negative F value. Although such a strategy guarantees to have valid service selections, it introduces significant computation overhead. In essence, many invalid SEPs are still generated and evaluated by the service selection algorithm. In contrast, the O-Decompose strategy eliminates all unnecessary computations. O-relations only store providers that offer the corresponding operations. In addition, the bitmap structure can help efficiently locate the valid service providers without carrying out expensive join operations. As a result, only valid SEPs are generated and evaluated by the service selection algorithms.

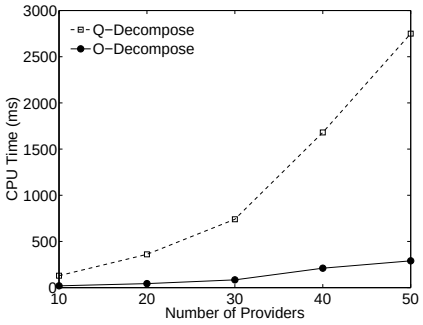

Figure 8. CPU Time Vs. Number of Providers

The results in Figure 9 further justify the above analysis. In this set of experiments, we set the number of providers in each service relation as 50. We vary the p value from 0.5 to 0.9. The performance of the Q-Decompose strategy does not change with p at all because all SEPs (valid and invalid) need to be evaluated. On the other hand, the performance of the O-Decompose strategy varies significantly with p. It offers an extremely efficient performance for small p values.

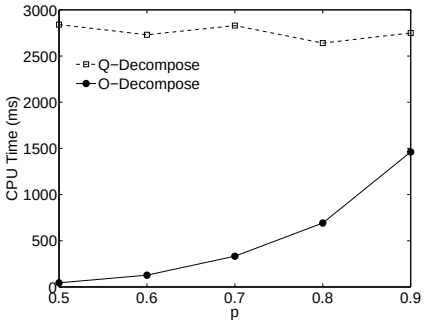

Figure 9. CPU Time Vs. p

5 Related Work

Service selection techniques have received considerable attention recently. In [15], a composite service optimization approach is proposed based on several quality of service parameters. Composite services are represented as a state-chart. The optimization problem is tackled by finding the best Web services to execute a composite service in the form of a linear programming problem. The service selection problem is investigated in [14] by using a combinato-

rial model and a graph model. Efficient algorithms are then designed to select the composite Web services. In [12], a formal service model is defined and then a dynamic programming based approach is proposed to select the best service providers. The service selection approach we presented in this paper relies on the same service model in [12]. It makes a key extension to these existing approaches by allowing service providers that partially match the predefined abstract service or composition plans. In addition, we present a systematic relational approach to efficiently select the best SEPs.

In [8], a Web Service Management System (WSMS) is proposed to enable optimized querying of Web services. A Web service $WS_i(\mathcal{X}_i^b, \mathcal{Y}_i^f)$ is modeled as a virtual table in the proposed WSMS. The values of attributes in $\mathcal{X}_i$ must be specified whereas the values of attributes in $\mathcal{Y}_i$ are retrieved. An algorithm is proposed to optimized access Web services. The optimization algorithm takes as input the classical Select-Project-Join queries over Web services. It arranges Web services in a query based on a cost model and returns an pipelined execution plan with minimum total running time of the query. The service selection algorithms proposed in this paper focus on the 'user-centered' QoWS. It facilitates service users in efficiently selecting the service providers with their best desired quality.

Several different service models have been developed for service composition or selection. OWL-S [7] specifies an ontology-based service model. It uses the concept of "IOPEs" to describe the behavior of Web services. It aims at setting up a framework for the automatic service discovery, invocation, composition and interoperation, and execution monitoring. In [2], Finite State Machines (FSMs) are adopted to model Web services (e-Services). A service model in this work consists of an external schema and an internal schema. The external schema is to specify the exported behavior of a service. The behavior is represented by a set of actions and the corresponding state transitions. The internal schema, on the other hand, specifies the information on which services execute each given action. The FSMs service model provided fundamental support for the service synthesis theory proposed in [2]. In [3], a service net is proposed to model Web services using Petri net. Based on the Petri-net service model, a service level algebra is proposed. The proposed algebra verifies the closure property. The algebra can be used to construct complex services by aggregating and reusing existing services. The reason why we adopt the service model presented in [12] is that it capture a key set of Web service features that are all essential for the service selection problem. The model allows us to develop a systematic relational approach to easily and efficiently select the best SEPs.

6 Conclusion

We present a service selection mechanism by using QoWS as a major differentiating criterion between different service providers. The selection mechanism is based on a formal service model that enables the usage of a relational database to store and retrieve QoWS information. We discuss two different ways about how to store the QoWS information from partially matching service providers. Q-relations treat these providers just like those providers that fully conform to the service schema but use some special place holders for the operations that they do not offer. The place holders guarantee that these providers will never be selected when a user request an operation that is not offered by them. O-relations do not require any place holder thus provide a more compact representation of the original service relation. However, expensive multi-table joins are required to identify the desired service providers if O-relations are used. We present a bitmap structure to resolve this issue and achieve efficient service selection by using O-relations.

References

[1] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-time monitoring of instances and classes of web service compositions. In *ICWS*, 2006.

[2] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *VLDB*, 2005.

[3] R. Hamadi and B. Benatallah. A petri net-based model for web service composition. In *Fourteenth Australasian database conference on Database technologies*, pages 191–200, 2003.

[4] R. Jurca, B. Faltings, and W. Binder. Reliable qos monitoring based on client feedback. In *WWW*, 2007.

[5] L. Mei, W. K. Chan, and T. H. Tse. An adaptive service selection approach to service composition. In *ICWS*, pages 70–77, 2008.

[6] M. Ouzzani and B. Bouguettaya. Efficient Access to Web Services. *IEEE Internet Computing*, 37(3), March 2004.

[7] OWL-S. http://www.daml.org/services/owl-s/, 2004.

[8] U. Srivastava, J. Widom, K. Munagala, and R. Motwani. Query Optimization over Web Services. In *VLDB*, 2006.

[9] S. Vinoski. Web services interaction models, part 1: Current Practice. *IEEE Internet Computing*, 6(3):89–91, 2002.

[10] Z. Xu, P. Martin, W. Powley, and F. Zulkernine. Reputation-enhanced qos-based web services discovery. In *ICWS*, pages 249–256, 2007.

[11] X. Ye and R. Mounla. A hybrid approach to qos-aware service composition. In *ICWS*, pages 62–69, 2008.

[12] Q. Yu and A. Bouguettaya. Framework for Web Service Query Algebra and Optimization. *ACM Trans. Web*, 2(1), 2008.

[13] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed. Deploying and managing web services: issues, solutions, and directions. *VLDB J.*, 17(3):537–572, 2008.

[14] T. Yu and K. Lin. Service selection algorithms for composing complex services with multiple qos constraints. In *ICSOC'05*, 2005.

[15] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng. Quality-driven Web Service Composition. In *WWW*, 2003.