Service Selection Algorithms for Web Services with End-to-end QoS Constraints*

Tao Yu and Kwei-Jay Lin
Dept. of Electrical Engineering and Computer Science
University of California, Irvine
Irvine, California 92697-2625, USA

Abstract

Web services are new forms of Internet software that can be universally deployed and invoked using standard protocol. Services from different providers can be integrated to provide composite services. In this paper, we study the end-to-end QoS issues of composite service by utilizing a QoS broker that is responsible for coordinating the individual service component to meet the quality constraint. We design the service selection algorithms used by QoS brokers to meet end-to-end QoS constraints. The objective of the algorithms is to maximize the user-defined utility while meeting the end-to-end delay constraint. We model the problem as the Multiple Choice Knapsack Problem (MCKP) and provide efficient solutions. The algorithms are tested for their performance.

1 Introduction

The Web services framework has evolved to become the software foundation for next generation enterprise and Web-based systems. By adopting standard-based protocols (such as SOAP, WSDL and UDDI), service components from different providers can be conveniently integrated into a *composite service* regardless of their locations, platforms and/or speed. As more and more composite Web services are deployed, many of them may share common services that are offered by some major service providers, such as stock services, search engines, etc. With growing and competing demands, popular service providers have started to offer different service levels so as to meet the needs of different user bases, by offering different service qualities based on user qualification or service cost. One example of such projects is the Oceano project at IBM [10].

The quality of a composite service is measured by its end-to-end quality, rather than the quality of any individual service component. A mechanism is needed to ensure that the end-to-end quality of a composite Web service is acceptable. Our research on the end-to-end QoS problem of composite service is based on the QoS-Capable Web Service architecture, QCWS[9], that defines a QoS broker between Web service clients and providers. In QCWS, every service provider is assumed to offer many service levels for the same service functionality. The QoS broker collects the QoS information of service providers (servers), makes service (and service level) selection decisions for clients, and then negotiates with servers to acquire QoS service commitments.

In this paper, we study the service selection algorithms used by QoS brokers. A broker receives service requests from clients and identifies services that may meet the functional needs of those requests as well as their QoS requirements. The service selection algorithm considers service cost, service response time, server load, and network delay to make the best selection that meets the end-to-end delay constraint. We model the problem as a Multiple Choice Knapsack Problem (MCKP) and show algorithms that can solve the problem efficiently. The algorithms may be adopted by QoS brokers to make dynamic run-time decisions.

The contribution of our research is as follows:

1. We define the QoS broker service for managing end-to-end QoS in composite Web services. The QoS broker service is different from, but may be integrated with, the process service in the BPEL process composition [11], the coordinator service in WS-Transaction [12], and/or the registry service for UDDI registries [13]. The QoS broker is designed to make service selection for client requests, based on their QoS constraints and requirements.

2. The objective function (called *utilities*) and constraint used by the algorithms are based on a rich set of system parameters, including static server information (service level), client QoS requirement (QoS

*This research was supported in part by NSF CCR-9901697.

Proceedings of the IEEE International Conference on E-Commerce Technology
0-7695-2098-7/04 $20.00 © 2004 **IEEE**

constraint), dynamic server capacity (service benefit), and network delay. They could also consider service reliability, availability, etc.

3. We present several service selection algorithms. For composite service that is structured as a pipeline or a DAG (Directed Acyclic Graph) and with only one QoS constraint (end-to-end delay in our study), we model the problem as a MCKP and identify a very efficient algorithm to solve it. To our knowledge, our work is among the first to model the end-to-end QoS problem as MCKP and presents efficient algorithms.

The rest of this paper is organized as follows. Section 2 provides a quick overview of Web service QoS parameters. Section 3 presents the end-to-end QoS system model. Various algorithms for the service selection problem are shown in Section 4. The review of some related work is given in Section 5. The paper is concluded in Section 6.

2 Web Service QoS Parameters

In our study, we consider the following QoS attributes as part of the Web service parameters.

- *Response time*: the amount of time to get a service request responded at the client side. This includes the total time for service and round-trip communication delay.
- *Service cost*: Some Web services are resource intensive. A Web service may bear different costs depending on the quality of the service requested.
- *Network delay*: the network transmission time required to receive the service. This is especially important for services with multimedia content such as video or graphics. The bandwidth attribute will also be important for Web service brokers to decide if a service should be invoked if the client is using a low bandwidth network such as wireless connection.
- *Service availability*: the probability that the service is available. This only measures the server availability in terms of responding to a request, not the result quality.

Since our goal is to build automated brokers for Web services, the QoS attributes that we consider are both easy to understand and to measure. These attributes can be collected on an automated system without any user intervention.

3 System Model

The main purpose of QoS broker is to help clients (1) finding qualified services and service levels; (2) selecting the most suitable one(s) in all candidates to fulfill its functional and QoS requirements. The service selection algorithm used by the QoS broker is very important in the QoS broker design. In this section, we present the system model of our proposed service selection algorithm.

3.1 Assumptions and definitions

Our system model and notations are defined as follows:

1. S: service class. A *service class* is a collection of individual Web services with a common functionality but different non-functional properties (e.g. costs, quality levels, etc.);
2. s: individual Web service in a service class, which resides on a specific server somewhere on the network;
3. l: one of the service levels provided by an individual Web service;
4. R: the end-to-end delay constraint.

Also we make the following definitions about a service:

1. For a service s, the number of service levels it provides is $L(s)$;
2. Each service level guarantees a service time $e(s,l)$, i.e., $e(s,l)$ is the maximum delay a client may experience by selecting service s at service level l;
3. Each service level has a maximum capacity $C_{max}(s,l)$, the maximum number of clients it can accept;
4. Each service level has a current capacity $C_{cur}(s,l)$, the current number of clients receiving service at this level;
5. Each service level has a fixed cost $c(s,l)$.

In our study, we define a *benefit* function based on the unused server capacity which is defined as $\frac{C_{max}(s,l)-C_{cur}(s,l)}{C_{max}(s,l)}$. The benefit function $b(s,l)$ is the benefit a client receives when selecting service s at level l. By sending new service requests to a server that has a lighter load, a client may get a faster response than running it on a busy server. Moreover, such decisions will distribute workloads more evenly among all servers and create a more globally balanced service system. The benefit function has the following properties:

Proceedings of the IEEE International Conference on E-Commerce Technology
0-7695-2098-7/04 $20.00 © 2004 **IEEE**

1. $b(s,l)$ is a continuous and increasing function of unused server capacity;
2. $b_{max}(s,l) = 1$, when $C_{cur}(s,l) = 0$;
3. $b_{min}(s,l) = 0$, when $C_{cur}(s,l) = C_{max}(s,l)$.

The specific benefit function can be defined freely as long as it satisfies the above characteristics. For example, a benefit function may be a linear function or exponential function of server capacity.

Since services may be provided by different providers, they communicate by passing requests and data on the network. Transferring results from one service to another or to the user incurs some delay. In this paper, we assume that the transmission delay and cost between any two services and between a service and a client are predefined and fixed. The transmission delay and cost between two services s_i and s_j are denoted as d_{ij} and c_{ij} . The transmission delay and cost from the last service s_i to the client are denoted as d_{iu} and c_{iu} . We include the network delay and cost in our algorithm.

One requirement of the service selection is that the service selection for a new request should not disturb the activities of existing clients. So the service and service level (s,l) selected by a broker must be among those currently available candidates i.e. $C_{max}(s,l) - C_{cur}(s,l) > 0$.

3.2 Utility Function

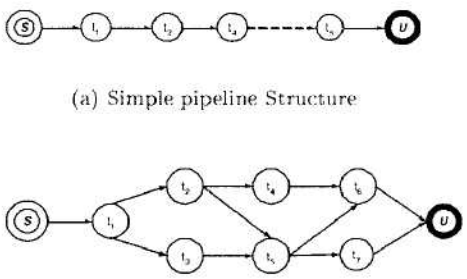
From the above definitions, we now present the objective function of the service selection problem. We call it the *utility function* $F(s,l)$. For convenience, $F(s,l)$ is also denoted as F_{sl} in this paper. Since a client wants to maximize the benefit it receives and minimize the cost it pays, we combine the benefit function $b(s,l)$ and cost function $c(s,l)$ together as the utility function F_{sl}:

$$F_{sl} = w_b * (\frac{b(s,l) - avgb}{stdb}) + w_c * (1 - \frac{c(s,l) - avgc}{stdc}) \quad (1)$$

where
w_b : Weight of the benefit, $0 < w_b < 1$
w_c : Weight of the cost, $0 < w_c < 1$ and $w_c = 1 - w_b$
$b(s,l)$: benefit of using service s at level l
$c(s,l)$: cost of using service s at level l
$avgb$: average benefit for all services and levels
$avgc$: average cost for all services and levels
$stdb$: standard deviation of all benefits
$stdc$: standard deviation of all costs

3.3 Single service vs. composite service

The Web services requested by a client can be divided into two categories. One is the single service, when a request can be accomplished by an individual service (e.g. when a user requests for a stock price). The other is the composite service, for which a request must be completed by a set of individual services together (e.g. when a user wants to make a travel plan that includes flight selection, hotel reservation, car rental, ticket purchase, etc.). In this paper, we consider the composite service with sequential execution only.



(a) Simple pipeline Structure

(b) DAG Structure

Figure 1: **Composite Service**

There are two types of composite service depending on how many execution paths a composite service has: *Simple pipeline* and *DAG structure* (Figure 1). Simple pipeline is the simplest composite service structure with only one execution path. All that a broker needs to do is to find a service (and its service level) in each service class along the execution path. DAG composite service has several execution paths (e.g. Figure 1(*b*) contains 5 possible paths). Each path can be considered as a simple pipeline composite service. Broker has to select an optimal execution path among the 5 paths.

4 Service Selection Algorithms

4.1 Single service

If only a single service is needed, the selection will choose the service with the lasrgest utility value. However, as mentioned before, we also need to consider the network delay in our model. After identifying the service class that can fulfill the user's request, we add the network transmission delay and cost between the user and a service to all service levels of that service. That is, $c(s_i,l) = c(s_i,l) + c_{iu}$and the response time $r(s_i,l) = e(s_i,l) + d_{iu}$. After the parameters are updated, a broker calculates the utility F_{sl} according to Eq. 1 for each candidate (service level and service) and selects the one that has the highest utility value.

Proceedings of the IEEE International Conference on E-Commerce Technology
0-7695-2098-7/04 $20.00 © 2004 **IEEE**

4.2 Pipeline composite service

For requests that must be fulfilled by a composite service, the QoS broker needs to decide how many single services are needed for this composite service. Then it needs to find the suitable service and service levels for each single service from the service class.

Again we must include the network delay in service selections. In this problem formulation, we assume the network connection overheads between any two services in the consecutive steps are the same. That is, if the data flow direction is $S_a \rightarrow S_b$, we assume the network overhead between any service s_i in S_a and any service s_j in S_b are all the same. The transmission delay and cost between any two services are then added to the sending service. That is, the transmission overhead from s_i to s_j will be added to s_i : $c(s_i, l) = c(s_i, l) + c_{ij}$ and $r(s_i, l) = e(s_i, l) + d_{ij}$. The transmission delay and cost between the last service and the client will be added to the last service.

For a pipeline composite service that has k steps (k service classes in an execution path) $(S_1, S_2, ..., S_k)$, suppose the total response time $\leq R$. The problem can be modeled as a MCKP. Given a set of items in several classes and a knapsack, where each item has a weight and profit, and the knapsack has a capacity, MCKP is to select one item from each class to be placed in the knapsack within the capacity yet has the highest total profit. We model the composite service selection problem as a MCKP in the following way:

1. The steps of the composite service represent the classes in MCKP;
2. Since each service in a service class has many service levels, each service level is a *candidate* for service selection; thus every candidate represents an item in that class in MCKP;
3. The response time of each candidate represents the weight of the item in MCKP;
4. The utility a candidate produces represents the profit of the item in MCKP;
5. The objective is to maximize the overall utility produced by the composite service under the constraint that the total response time $\leq R$;

The problem is thus formulated as:

$$
\begin{aligned}
Max \quad & \sum_{i=1}^{k} \sum_{j \in S_i} F_{ij} x_{ij} \\
Subject\ to \quad & \sum_{i=1}^{k} \sum_{j \in S_i} r_{ij} x_{ij} \leq R \\
& \sum x_{ij} = 1,\ i = 1, ..., k,\ j \in S_i \\
& x_{ij} \in \{0, 1\},\ i = 1, ..., k,\ j \in S_i \qquad (2) \\
F_{ij}: \quad & Utility\ value\ at\ step\ i\ for\ candidate\ j \\
r_{ij}: \quad & Response\ time\ of\ candidate\ j\ at\ step\ i \\
R: \quad & Total\ response\ time
\end{aligned}
$$

The MCKP problem is NP-hard. In the following we present three algorithms that can be used: exhaustive search, dynamic programming and a minimal algorithm for MCKP.

Before using any algorithm to do the service selection, some preprocessing on the candidates of each class may reduce the number of candidates in each class. The following criteria is presented in [1]:

If two items a and b in the same class S_i, satisfy

$$r_{ia} \leq r_{ib} \text{ and } F_{ia} \geq F_{ib} \qquad (3)$$

then an optimal solution to MCKP with $x_{ib} = 0$ exists. We thus can delete item b from the candidate list in S_i.

Exhaustive search algorithm

This algorithm is a straightforward one by constructing all service combinations and compares their utilities. It can always produce the optimal solution but is time and memory consuming. So it is only suitable for when the number of classes and the items of each class are all small. For a composite service contains k steps and step i has $L(i)\,(i = 1, 2, ..., k)$ candidates, the time complexity of the exhaustive search algorithm is $O(\prod_{i=1}^{k} L(i))$.

Dynamic programming algorithm

The MCKP problem can be solved in pseudo-polynomial time through dynamic programming. Given a pair of integers $l\,(1 \leq l \leq k)$ and $\hat{c}\,(0 \leq \hat{c} \leq R)$, consider the sub-instance of MCKP consisting of subsets $S_1, S_2, ... S_l$ and capacity $\hat{c}$. Let $f_l(\hat{c})$ denotes its optimal solution value. The problem can be solved by the following dynamic programming formulation:

Proceedings of the IEEE International Conference on E-Commerce Technology
0-7695-2098-7/04 $20.00 © 2004 **IEEE**

$$Let \quad \bar{r} = min\{r_j,\ j \in S_i\} \quad i = 1, 2, ..., k$$

$$f_1(\hat{c}) = \begin{cases} -\infty & \hat{c} = 0, 1, ..., \bar{r_1} - 1 \\ max\{F_{1j} : j \in S_1, r_j \le \hat{c}\} & \hat{c} = \bar{r_1}, ..., R \end{cases}$$

$$f_l(\hat{c}) = \begin{cases} -\infty & \hat{c} = 0, 1, ..., \sum_{k=1}^{l} \bar{r_k} - 1 \\ & (2 \le l \le k) \\ max\{f_{l-1}(\hat{c} - r_j) + F_{lj} : j \in S_l, r_j \le \hat{c}\} & \\ \hat{c} = \sum_{k=1}^{l} \bar{r_k}, ..., R & \end{cases} \tag{4}$$

The optimal solution is the state corresponding to $f_k(R)$. For a composite service contains k steps and step i has $L(i)$ $(i = 1, 2, ..., k)$ candidates, the time complexity of the dynamic programming algorithm is $O(\sum_{i=1}^{k} L(i) * R)$.

Pisinger's Algorithm

In [1], David Pisinger introduces an algorithm for efficiently solving the MCKP problem. This algorithm first solves the linear MCKP (LMCKP) problem by using a partitioning algorithm and derives an initial feasible solution (*initial core*) to MCKP. It then uses dynamic programming to expand the initial core by adding new classes as needed. In this algorithm, a minimum number of classes are considered to solve MCKP and it uses the minimum effort for sorting and reduction.

1. Solving Linear Multiple-Choice Knapsack Problem (LMCKP)

   In Eq. (2), if we relax the integrity constraint $x_{ij} \in \{0, 1\}$ *to* $0 \le x_{ij} \le 1$, the problem becomes LMCKP. In [2] and [3], Zemel and Dyer each developed linear time algorithms for LMCKP. Both algorithms are based on the convexity of the LP-dual problem to Eq. (2). For the dual problem, we can pair the dual line segments and delete the unpromising ones according to some dominance criteria.

   Based on Dyer and Zemel's algorithms, [1] presents a partitioning algorithm to solve the LMCKP. The optimal solution $x^*$to LMCKP is composed by the LP-optimal choices b_i in each class, where $x_{ib_i} = 1$. One of the classes S_a may contain two non-zero fractional variables x_{ab_a} and $x_{ab'_a}$ $(x_{ab_a} + x_{ab'_a} = 1)$. If x^* has no fractional variables, it is already the optimal solution to MCKP. Otherwise, the fractional class S_a is defined as the *initial core* for MCKP. We then continue the following step.

2. Solving MCKP

   Given an initial core and the set of $\{b_i | i \ne a\}$ from step 1, the positive and negative gradient λ_i^+ and λ_i^- for each class $S_i, i \ne a$ are defined as:

$$\lambda_i^+ = \max_{j \in S_i,\, r_{ij} > r_{ib_i}} \frac{F_{ij} - F_{ib_i}}{r_{ij} - r_{ib_i}}, i = 1, 2, ..., k, i \ne a, \tag{5}$$

$$\lambda_i^- = \max_{j \in S_i,\, r_{ij} < r_{ib_i}} \frac{F_{ib_i} - F_{ij}}{r_{ib_i} - r_{ij}}, i = 1, 2, ..., k, i \ne a, \tag{6}$$

   We then sort the sets $L^+ = \{\lambda_i^+\}$ in decreasing values, and $L^- = \{\lambda_i^-\}$ in increasing values. Starting from the initial core, we will expand the core by alternately including a new (not yet selected) class S_i that has the largest λ_i^+ from L^+ or the smallest λ_i^- from L^-;

   Before adding a new class to the core, an upper bound test (explained in [1]) can be used to fathom unpromising items from the class. If only one item is left, the class may be excluded from further selection in L^+ and L^-. Otherwise the class is added to the core.

   In a core including classes $C = \{S_{r_1}, ..., S_{r_m}\}$, the set of partial vectors is given by $Y_C = \{(y_1, ..., y_m) \,|\, y_i \in \{1..n_{r_i}\},\ i = 1..m\}$. Each variable y_i determines $x_{iy_i} = 1$ in class S_i, while $x_{ij} = 0, j \ne y_i$. Moreover, the vector $\bar{y_i} = (y_1, ..., y_m) \in Y_C$ will result in a state (μ_i, π_i, v_i), where μ_i, π_i are defined by Eq. (7) and (8) below, while v_i is just a convenient representation of $\bar{y_i}$.

$$\mu_i = \sum_{S_i \in C} r_{iy_i} + \sum_{S_i \notin C} r_{ib_i} \tag{7}$$

$$\pi_i = \sum_{S_i \in C} F_{iy_i} + \sum_{S_i \notin C} F_{ib_i} \tag{8}$$

   After adding a new class in the core, some of the states in the core can be fathomed by another upper bound test [1]. When all classes have been selected, the optimal solution for MCKP can be found.

The computational experiments in [1] show that this algorithm is very efficient and much faster than the dynamic programming algorithm. For a composite service contains k steps (k classes, $S_1, .., S_k$) and step i has $(L(i)\,(i = 1, 2, ..., k)$ candidates, each class S_i in the core C has n_i the time complexity of Pisinger's algorithm is : $O(\sum_{i=1}^{k} L(i) + R * \sum_{S_i \in C} L(i))$.

In the worst case, the time complexity of Pisinger's algorithm is the same as dynamic programming for MCKP. However, as we will show later in this paper, Pisinger's algorithm usually converges very fast, allowing its computation time to be much less than dynamic programming.

Proceedings of the IEEE International Conference on E-Commerce Technology
0-7695-2098-7/04 $20.00 © 2004 **IEEE**

4.3 DAG composite service

Composite services that are structured as DAG present a more challenging service selection problem since there are many possible execution paths. Two approaches may be used to find the optimal execution path in a DAG composite service.

1. Find all possible execution paths. For each path, treat it as a pipeline composite service and find the highest overall utility (using Pisinger's algorithm). Compare the overall utility values produced for all paths and choose the one with the highest value as the optimal service selection.
2. Use Constrained Bellman-Ford(CBF [8]) to find the optimal path that produces the highest overall utility. In the algorithm, instead of searching for the shortest path, we search for the highest utility path. Detailed description of CBF can be found in [8].

Due to the space constraint, we will not compare these two approaches in this paper.

4.4 Composite service example

Now we present an example to show the procedure of Pisinger's algorithms used for a pipeline composite service. The example is defined as follows:

- The benefit function $b(s,l)$ is defined as:

$$b(s,l) = \frac{1-e^{-\frac{C_{max}(s,l)-C_{cur}(s,l)}{C_{max}(s,l)}}}{1-e^{-1}} \qquad (9)$$
$$0 \le C_{cur}(s,l) \le C_{max}(s,l)$$

- Constraint: the end-to-end response time $R \le 61$;
- The composite service contains 4 sequential services ($k = 4$), each has 4 service levels ($n = 4$). We assume that all levels are available, *i.e.* $C_{max}(s,l) - C_{cur}(s,l) > 0$. The response time, cost, maximum capacity and current capacity of each service and service level are shown in Tables 1,2 and 3:

According to the definition of the benefit function Eq. (9) and utility function Eq. (1), we can get the utility of each service level. Then we convert the utility to non-negative integers using the formula $F(s,l) = floor(F(s,l) * 200) + min(F(s,l))$. The converted utility is shown in Table 4. Also, we use the preprocessing criteria 3 to delete some items in each class and produce the final $r(s,l)$ & $F(s,l)$ shown in Table 5.

When using Pisinger's algorithm, we first solve the corresponding LMCKP to get the initial solution to

Response time $r(s,l)$				
	Level 1	*Level 2*	*Level 3*	*Level 4*
Service 1	3	12	21	30
Service 2	4	10	18	26
Service 3	7	18	30	36
Service 4	9	17	24	33

Table 1: Response time

Cost $c(s,l)$				
	Level 1	*Level 2*	*Level 3*	*Level 4*
Service 1	33	28	19	13
Service 2	24	19	14	8
Service 3	25	20	15	10
Service 4	2)	21	16	11

Table 2: Cost

Response time (cost) $r(s,l)(c(s,l))$				
	Level 1	*Level 2*	*Level 3*	*Level 4*
Service 1	3 (33)	12 (28)	21 (19)	30 (13)
Service 2	4 (24)	10 (19)	18 (14)	26 (8)
Service 3	7 (25)	18 (20)	30 (15)	36 (10)
Service 4	9 (26)	17 (21)	24 (16)	33 (11)

Table 3: Max (Current) capacity

Utility $F(s,l)$				
	Level 1	*Level 2*	*Level 3*	*Level 4*
Service 1	10	167	191	162
Service 2	10	140	232	240
Service 3	3	18	136	241
Service 4	16	143	143	96

Table 4: Utility

Response time $r(s,l)$[Utility $F(s,l)$]				
	Level 1	*Level 2*	*Level 3*	*Level 4*
Service 1	3[10]	12[167]	21[191]	-
Service 2	4[10]	10[140]	18[232]	26[240]
Service 3	7[3]	18[18]	30[136]	36[241]
Service 4	9[16]	17[143]	-	-

Table 5: Response time [Utility] after preprocessing

Proceedings of the IEEE International Conference on E-Commerce Technology
0-7695-2098-7/04 $20.00 © 2004 **IEEE**

L^+		
No.	*dr[dF]*	*Class{Items(r[F])}*
1	*9[24]*	$S_1\{12[167], 21[191], 3[10]\}$
2	*8[8]*	$S_2\{18[232], 26[240], 4[10], 10[140]\}$
3	*1[0]*	$S_4\{17[143], 9[16]\}$

Table 6: L^+set

L^-		
No.	*dr[dF]*	*Class{Items(r[F])}*
1	*8[92]*	$S_2\{18[232], 26[240], 4[10], 10[140]\}$
2	*8[127]*	$S_4\{17[143], 9[16]\}$
3	*9[157]*	$S_1\{12[167], 21[191], 3[10]\}$

Table 7: L^-set

MCKP Eq. (10). The initial core contains items: $\{7[3], 18[18], 30[136], 36[241]\}$.

$$\begin{array}{llll} S_1: & b_1 = 2, & r_{1b_1} = 12, & F_{1b_1} = 167; \\ S_2: & b_2 = 3, & r_{2b_2} = 18, & F_{2b_2} = 232; \\ S_3: & fractional & class, & initial\,core \\ S_4: & b_4 = 2, & r_{4b_4} = 17, & F_{4b_4} = 143. \end{array} \quad (10)$$

Starting from the initial core S_3, we calculate the positive and negative gradient $\lambda^+$and $\lambda^-$for each class S_i, $i \neq 3$ and sort $L^+ = \{\lambda_i^+\}$ in decreasing values (Table 6), and $L^- = \{\lambda_i^-\}$ in increasing values (Table 7).

The set of partial vectors $Y_C = \{(\mu, \pi_i, \upsilon_i)\}$ in the initial core C has 4 states: $Y_C = \{(54, 545, 0), (65, 560, 1), (77, 678, 2), (83, 783, 3)\}$. After doing the state reduction according to the upper bound test (see [1]), Y_C becomes $Y_C = \{(54, 545, 0), (65, 560, 1)\}$. In this Y_C , $(54, 545, 0)$ is chosen with item (7[3]) and utility $z = 545$. In the following steps, we will add new classes to the core until all classes have been considered.

1. Add class $S_1\{12[167], 21[191], 3[10]\}$ from L^+ to the core, $Y_C = \{(54, 545, 0), (63, 569, 1)\}$; The item chosen in S_1 is 12[167], the utility $z = 545$;

2. Add class $S_2\{18[232], 26[240], 4[10], 10[140]\}$ from $L^-$to the core, $Y_C = \{(54, 545, 1), (63, 569, 3)\}$; the item chosen in S_2 is 18[232], the utility $z = 545$;

3. Add class $S_4\{17[143], 9[16]\}$ from $L^-$to the core, $Y_C = \{(63, 569, 3)\}$; The item chosen in S_4 is 17[143], the utility $z = 545$;

At this point, the core is complete. The optimal solution includes the chosen item in each class: $\{12, 18, 7, 17\}$ and the maximum utility value is $z = 545$.

		Running time (μs)		
Service (k)	*Cand. (n)*	*Exhaustive Search*	*Dynamic Programming*	*Pisinger's*
5	*5*	*127*	*99*	*56*
5	*50*	-	*4649*	*116*
6	*5*	*1131*	*130*	*37*
10	*10*	-	*1395*	*88*
10	*100*	-	*35668*	*217*
10	*1000*	-	*778990*	*1701*
20	*100*	-	*289071*	*304*
20	*1000*	-	*7 s 67617 μs*	*3240*
50	*100*	-	*1 s 830409 μs*	*685*
50	*1000*	-	*26 s 928142 μs*	*4778*
100	*100*	-	*6 s 273286 μs*	*1294*

Table 8: Running time comparison

4.5 Running Time Experiments

We have conducted many tests using the three algorithms to compare their running time. Table 8 shows the test results. Since the exhaustive search algorithm is time and memory consuming, it is only suitable for the situation when k (service stages of the composite service) and n (number of candidates in each stage) are both small (In our tests, we choose $k + n \leq 25$). When k and n become larger, this algorithm quickly ran out of memory in our experiments. Between dynamic programming and Pisinger's algorithm, Pisinger's algorithm is much faster and more efficient than dynamic programming. For example, when $k = 10$ and $n = 100$, it takes dynamic programming $35,668\mu s$ to finish the computation while Pisinger's algorithm only needs $217\mu s$.

The test result shows that Pisinger's algorithm is the best algorithm for service selection in composite service, especially when the number of candidates in each service is large.

5 Related Work

The end-to-end Web services selection is part of the Web service composition problem. Many have worked on this topic. In [7], authors present a framework SELF-SERV for declarative Web services composition using statecharts. The resulting services can be executed in a decentralized way in a dynamic environment. Their service selection approach uses a local selection strategy. The selection of a service is determined independently to other tasks of the composite services. Although the service selection is locally optimal, they may not satisfy the global constraint such as the end-to-end delay.

The closest work to ours is [6] which gives a quality

Proceedings of the IEEE International Conference on E-Commerce Technology
0-7695-2098-7/04 $20.00 © 2004 **IEEE**

driven approach to select component services during the execution of a composite service. They consider multiple QoS criteria such as price, duration, reliability and take into account of the global constraint, as the end-to-end delay constraint presented in our paper. One difference between our models is that they don't have the service level concept. In their work, a service can only have one processing time and one cost. Their solution to the service selection problem is to use the linear programming technique, which is usually too complex for run-time decisions.

[5] describes a distributed QoS management architecture for a complex distributed real-time system that accommodates and manages different dimensions and measures of QoS. It is based on a flexible computation approach to trade the amount of time and resources used to produce different qualities of result. Our work is different from theirs because the quality provided by each service is from a third party (service provider). So we can only select the quality of service among existing service levels instead of adjusting the system resource for each service.

6 Conclusions

In this paper, we study the Web Service end-to-end QoS constraint issue by utilizing a QoS broker that is responsible for coordinating among individual service components to meet the quality constraint for client. We have presented service selection algorithms. The objective of the algorithms is to maximize user-defined service utilities while meeting the end-to-end delay constraint. The service selection problem is modeled as a MCKP and efficient algorithms are presented to solve it. We have studied the performance of several algorithms and shown that Pisinger's algorithm has a very efficient run time even for problems that have a fairly large data set. We believe the proposed model and algorithm present a practical solution to the end-to-end QoS guarantee problem for composite Web services.

References

[1] Pisinger, D., "A minimal algorithm for the Multiple-choice Knapsack Problem", European Journal of Operational Research, 83, 394-410 ,1995.

[2] Zemel, E., "An O(n) algorithm for the linear multiple choice knapsack problem and related problems", Information Processing Letters, 18, 123-128, 1984.

[3] Dyer, M.E., "An O(n) algorithm for the multiple-choice knapsack linear program", Mathematical Programming, 29, 57-63, 1984.

[4] Martello, S., & Toth, P. "Knapsack Problems, Algorithms and Computer Implementations", John Wiley & Sons Ltd. 1990.

[5] Shankar,M., DeMiguel, M. and Liu, J. W.S., "An end-to-end QoS management architecture", In Proc. 5th Real-Time Technology and Applications Symposium, 1999.

[6] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. Z.. "Quality Driven Web Service Composition". In Proc. 12th International World Wide Web Conference (WWW), 2003.

[7] Benatallah, B., Dumas, M., Sheng, Q.Z., and Ngu, A., "Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services." In Proc. of IEEE ICDE'02, pp. 297-308, San Jose, 2002

[8] Widyono, R., "The design and evaluation of routing algorithms for real-time channels". Tech. Rep. TR-94-024, University of California at Berkeley, International Computer Science Institute, June 1994.

[9] Chen, H., Yu, T. and Lin, K.J., "QCWS: An Implementation of QoS Capable Multimedia Web Services". In Proc. of IEEE 5th Int. Symp. Multimedia Software Engineering, Taiwan, Dec 2003.

[10] The Oceano project, IBM, http://www.research.ibm.com/oceanoproject/index.html.

[11] Business Process Execution Language for Web Services, http://www.ibm.com/developerworks/library/ws-bpel1/.

[12] Web Services Transaction (WS-Transaction) http://www.ibm.com/developerworks/library/ws-transpec/

[13] OASIS Universal Description, Discovery and Integration Specification TC, http://www.oasis-open.org/committees/uddi-spec/.

Proceedings of the IEEE International Conference on E-Commerce Technology
0-7695-2098-7/04 $20.00 © 2004 **IEEE**