A Hybrid Approach to QoS-aware Service Composition

Xinfeng Ye Rami Mounla
Department of Computer Science, Auckland University, New Zealand
xinfeng@cs.auckland.ac.nz rami@cs.auckland.ac.nz

Abstract

QoS-aware service composition intends to maximize the QoS of a composite service when selecting service providers. This paper proposes a service composition scheme that uses a combination of Integer Programming, case-based reasoning, and, genetic algorithms techniques. The scheme reduces the service composition costs by reusing existing compositions. Experiments show that, compared with solutions purely based on Integer Programming, the proposed scheme is effective in reducing the time for carrying out service composition.

1. Introduction

A *composite service* consists of many tasks. The main functionalities of the tasks can be provided by Web Services. A *service composition* means finding a service provider for each of the tasks in a composite service. As many service providers might provide services with identical or overlapping functionality, multiple candidates might be available for providing the functionality of a task in a composite service. The *quality of service* (QoS) of these candidate providers can be used to distinguish these service providers. QoS is the non-functional property of a web service. It is a combination of quality properties of a web service, such as response time, throughput, availability, reliability, cost etc. For a composite service, apart from the functional requirements for each of the tasks in the service, a user might also specify some QoS constraints on the composite service, e.g. response time, cost etc. Therefore, when solving a service composition problem, apart from matching the functionality of the tasks with the candidate providers, it is also necessary to ensure that the QoS constraints on the composite service can be satisfied.

Many researches have been carried out on QoS-aware service composition [17, 18]. Many proposed schemes use a function to calculate a score that represents the QoS of a composite service. During the service composition, these schemes attempt to maximize the QoS score of the composite service. Many schemes model the service composition problem as a multiple-choice knapsack problem and solve the problem using the Integer Programming technique. For a complex composite service, it might take a while to find a solution. The cost of finding a solution can be reduced if previous solutions can be re-used for future requests from the users. However, different users might have different preferences to the QoS properties of the services. For example, for two users that use the same composite service, one might prefer to keep the cost of running the service low while the other one might prefer to minimize the response time regardless the cost of running the service. Thus, a composition that produces a high QoS score for one user might not have the same score for another user. In addition, simply reusing the previous solutions in the presence of the QoS fluctuation of the service providers might result in a suboptimal service composition being used. This is because, if the QoS of a service provider in a previous solution deteriorates, the QoS score of the composite service that uses the solution might be worse than using a composition in which an alternative service provider is used. On the other hand, if the algorithm for solving the service composition problem is executed for each user's request, the composition cost might become too high.

This paper proposes a QoS-aware service composition scheme. The objective of the scheme is to reduce the service composition costs for complex composite services. The scheme uses a combination of Integer Programming (IP), case-based reasoning (CBR), and, genetic algorithms techniques. It reduces the composition cost by reusing the existing solutions. Instead of simply reusing previous solutions, the techniques in case-based reasoning are used to find existing solutions that are similar to the problem to be solved. To cope with the fluctuation of the QoS of the service providers, a genetic algorithm (the roulette-wheel selection) is used for selecting an existing solution. Some experiments were carried out to compare the performance of the proposed scheme with

978-0-7695-3310-0/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWS.2008.29

an approach purely based on IP. Experiments are also used to show that the scheme can adapt to the QoS fluctuations of the service providers.

This paper is organized as follows. §2 explains the background information about the techniques used in this paper. §3 discusses the related works. §4 describes the details of the proposed scheme. §5 shows the data collected from some experiments of the proposed scheme. Conclusions and future works are given in §6.

2. Background

2.1. QoS-aware Service Composition

An *execution plan* of a composite service is a set of service providers that are used to provide the functionalities of the tasks in the composite service. The QoS of a composite service can be derived from the QoS of the service providers in the execution plan of the composite service. For example, the cost of a composite service is the sum of the cost of all the service providers in the execution plan of the composite service. As there might be multiple candidates for each task in a composite service, a composite service might have several execution plans.

The Multiple Criteria Decision Making and Simple Additive Weighting techniques [6] are commonly used in comparing the QoS of different services [18]. In these techniques, a user assigns a weight to each of the QoS properties according to the user's preference. For example, if a user regards the response time is twice as important as the cost, the user can assign 2 as the weight for the response time and 1 as the weight for cost. Then, the QoS values are normalized by scaling them to a value within [0, 1]. If the maximum and the minimum value of a QoS property are equal, the scaled value is set to 1. Otherwise, a QoS value is scaled using the following formulas. In the formulas, V^{max} and V^{min} are the maximum and the minimum value of the property. V is the QoS value to be scaled.

$$SV = \frac{V^{\max} - V}{V^{\max} - V^{\min}} \qquad (1)$$

$$SV = \frac{V - V^{\min}}{V^{\max} - V^{\min}} \qquad (2)$$

Formula (1) is used to scale the negative property, i.e. the higher the value, the lower the quality, e.g. the response time, the execution cost, etc. Formula (2) is used to scale positive attribute, i.e. the higher the value, the higher the quality, e.g. reliability, etc.

The QoS score of a service is calculated according to the formula below. In the formula, PS denotes the set of QoS properties of a service, W_i is the weight assigned to property i, and, SV_i is the scaled value of property i.

$$Score = \sum_{i \in PS} (SV_i * W_i) \qquad (3)$$

In QoS-aware service composition, the objective is to find an execution plan that could maximize the QoS score of a composite service while satisfying the QoS constraints given by the users. The service composition problem can be solved using IP. To use an IP solver, an objective function and a set of constraints need to be specified. For service composition, the objective function should be $max(Score)$ (i.e. maximize the QoS score of a composite service). The constraints specify (a) the service providers in the solution should satisfy the QoS constraints given by the user, (b) only one service provider is selected for each of the tasks in a composite service, and, (c) the selected tasks should be executed in the order specified in the composite service.

2.2. Case-based Reasoning

Case-based reasoning (CBR) is an interesting method to use when facing a problem that is expensive and time consuming to solve. A CBR system solves new problems based on the solutions for similar problems. Given a new problem, the system retrieves the solution for a similar problem, alters it accordingly, uses it, and stores the new solution for future reference. When retrieving a previous solution, similarity functions are used for identifying the previously encountered problems that are similar to the current problem. The similarity functions are domain specific.

2.3. Genetic Algorithm

A Genetic Algorithm is a search technique inspired by natural genetic evolution. It starts with an initial population. Each individual in a population has a fitness level which may help measuring its probability for being selected by the search algorithm. The fitness of an individual is calculated using a fitness function. In order to select an individual, the fitness values of the individuals need to be normalized and ranked. Several techniques, e.g. roulette-wheel selection, linear-rank selection, etc., can be used to select an individual.

The roulette-wheel selection (RWS) is a commonly used selection technique. With the RWS, the probability of an individual being selected is proportional to its fitness. Thus, if f_i represents the fitness of an individual i in a population with n individuals, the probability of i being selected is $p_i = \frac{f_i}{\sum_{i=1}^{n} f_i}$. Using the RWS, the fitter individuals are more likely to be chosen while the weak ones also have a chance to be selected. It is important to give the weak

individuals a chance to be selected, since an individual might have been wrongly classified as weak [14].

3. Related Works

QoS-aware service composition has been studied by many researchers. Jaeger et al. [7] discussed how to aggregate the QoS properties of composite services. Zeng et al. [18] presented an approach for solving the service composition problem using local and global optimization. It pointed out that local optimization might not result in a composition that maximizes the QoS score of a composite service. Zeng et al. uses IP to carry out global optimization. [17] studied two approaches for service selection: the combinatorial approach and the graph approach. Like [18], they also use IP in their solutions. Thiβn et al. [15] proposed a service composition scheme that uses a step-wise merging approach to reduce the number of paths to be considered. Li et al. [11] described a scheme for constructing service overlay networks. They modeled the problem as a single-constrained problem and solved the problem using Dijistra's shortest path algorithm. Thiβn's and Li's approach are both similar to the local optimization approach in [18]. Unlike the scheme in this paper, none of the above-mentioned approaches attempt to solve the service composition problem by re-using the previous solutions.

Limthanmaphon et al. [12], Cheng et al. [1] and Lajmi et al. [10] proposed service composition schemes using the CBR techniques. They used the CBR techniques to find the similarities in the activities of the services. Their use of the CBR techniques focuses on service discovery. Unlike these approaches, the scheme in this paper focuses on QoS-aware service composition.

Howard et al. [5] proposed a brokering system that used the KDSWS system [4] for discovering, selecting, and, configuring Semantic Web Services. Their focus was around semantic capabilities, brokering system to execute composite Web services, etc. Unlike the work in this paper, their approach is not QoS-oriented, and does not optimize the service composition globally.

Thio et al. [16] stated the drawback of using Service Level Agreement management frameworks. They proposed a mechanism for monitoring the QoS properties on both clients and service providers. The collected QoS values are stored in an UDDI registry and will be used for finding services that satisfy clients' QoS requirements in the future. Different to [16] that did not optimize the plan in terms of the QoS score of the service, the scheme in this paper uses both CBR and IP to find and optimize the plan that satisfies a client's requirement.

Day et al. [2] proposed a composition scheme based on a rule-based system JESS [3]. In their approach, the rule-based system decides the best candidate that satisfies the client's request according to the values of the QoS properties stored in a forum. Unlike the scheme in this paper, [2] only considers finding a service provider for a single task while the scheme in this paper focuses on finding an execution plan for a composite service.

4. The Scheme

4.1. An Overview of the Scheme

The system consists of several entities as shown in Figure 1. The UDDI registry records all the available services. It is assumed that the service providers use the same ontology to describe their services when registering with the UDDI registry. As the scheme in [16], the QoS properties of the services are also recorded in the UDDI registry. The values of theses properties are provided by the service providers.

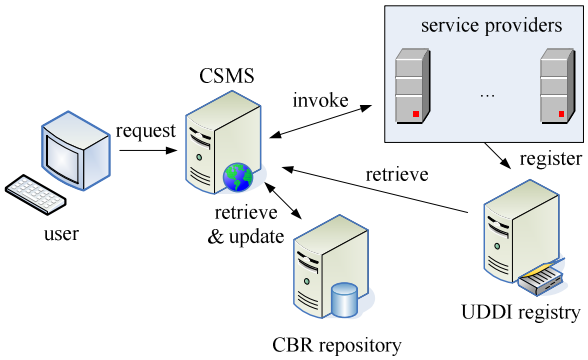


Figure 1 A Conceptual Diagram of the Scheme

Each concrete execution of the execution plan of a composite service is called an *instance* of the plan. The CBR repository stores the instances of the execution plans of the composite services. The Composite Service Management System (CSMS) is responsible for producing execution plans for composite services. The CSMS produces the execution plans either by retrieving suitable plans from the CBR repository or by using an IP solver to create a plan when no suitable plans exist in the CBR repository. Plans generated by the IP solver are stored in the CBR repository for possible future use. When a user wants to execute a composite service, the user sends a request to the CSMS. The CSMS first checks the CBR repository to find out whether there are existing execution plans that match the user's request. If there are matching plans, the CSMS chooses one of the plans to carry out the

user's request. Otherwise, the CSMS uses an IP solver to generate a new plan.

The CSMS also acts as the execution engine. When an execution plan is chosen for a composite service, the CSMS (a) invokes the service providers in the plan in the order specified by the composite service, and, (b) monitors the QoS properties of the services. The CSMS records the QoS values of each instance of a plan in the CBR repository.

4.2. Matching Instances of Execution Plans

In case-based reasoning, an existing solution should have some similarity with the problem that is being solved in order to be reused. The CSMS considers three properties of a composite service when matching it with existing instances of plans. The three properties are: the workflow of the composite service, the time at which the composite service is to be executed, and, the weights being assigned to the QoS properties when computing the QoS score.

The workflow of a composite service specifies what services (i.e. tasks) constitute the composite service and in which order these services are executed. Thus, the workflow defines a composite service. Hence, to match with a composite service that a user wants to execute (the composite service is denoted as *CS* in the following discussion), an existing plan must have exactly the same workflow as the *CS*. It is assumed that the users use the same ontology when describing their services. Thus, checking whether two composite services are the same is to check whether they contain the same set of tasks and whether the tasks are combined together using the same workflow pattern. To make the matching of the workflows of the composite services efficient, the SHA1 hash function is used to generate the digest of the BPEL file describing the workflow of a composite service. Thus, by comparing the digests of the workflows of two composite services, we can decide whether the two workflows are identical.

The time at which a service is executed is also important since the QoS of a service might fluctuate during the day. For example, at night time, the QoS of a service might be better than during normal business hours due to a lower system load. The matching for the execution timing does not need to be precise. Function $time_{sim}$ is used to measure the similarity between an instance of an execution plan and the *CS* in terms of execution time. In $time_{sim}$, t_{plan} is the time at which an existing plan was executed and t_{cs} is the time at which the *CS* is to be executed.

$$time_{sim}(t_{plan},t_{cs})=\begin{cases}0 & \text{if } t_{plan}=nil\\ |t_{plan}-t_{cs}| & \text{if } |t_{plan}-t_{cs}|\le 12\\ 24\bmod(|t_{plan}-t_{cs}|) & \text{otherwise}\end{cases}$$

A smaller value of $time_{sim}$ means a higher similarity between an existing plan and the *CS*. For plans that are generated by the IP solver, if they have not been executed, t_{plan} does not exist (i.e. $t_{plan} = nil$). To allow this kind of plans to have a chance to be selected for execution, their $time_{sim}$ are set to 0. The modulo operation ensures that the difference between t_{plan} and t_{cs} can be calculated correctly. For example, assume that t_{plan} is 1:00 and t_{cs} is 22:00. The difference between the two should be 3 hours instead of 21 hours.

The weights assigned to different QoS properties when using formula (3) to calculate the QoS score of a service are also important as they affect the final QoS score of a given execution plan. Function $weight_{sim}$ is used to determine the similarity between two sets of weights. In $weight_{sim}$, *PS* is the set of QoS properties, SV_i is the scaled QoS value of the i^{th} QoS property, and, WS_{plan} and WS_{CS} denote the set of weights assigned to the QoS properties by an existing plan and *CS* respectively.

$$weight_{sim}(WS_{plan},WS_{CS})=\frac{diff}{\sum_{i\in PS,W_i\in WS_{plan}}(SV_i*W_i)}$$

where

$$diff=\left|\sum_{i\in PS,W_i\in WS_{plan}}(SV_i*W_i)-\sum_{i\in PS,W_i\in WS_{CS}}(SV_i*W_i)\right|$$

For both $time_{sim}$ and $weight_{sim}$, the smaller of their values, the higher similarity between the existing plan and the *CS*. Thresholds ε_{time} and ε_{weight} are set such that an existing plan is regarded as similar to the *CS* if the following condition holds. In the expression, function *digest* gives the digest of a workflow according to the SHA1 function:

$$(digest(plan)=digest(CS))\wedge(time_{sim}(t_{plan},t_{cs})\le\varepsilon_{time})\\ \wedge(weight_{sim}(WS_{plan},WS_{CS})\le\varepsilon_{weight})$$

The instances of the execution plans stored in the CBR repository comply with Kolodner's work [9]. Each instance consists of three elements:

- Problem: the workflow of a composite service and the SHA1 digest of the workflow
- Solution: the execution plan of the composite service
- Evaluation: the values of the QoS properties, the time at which the plan is executed, and, the weights assigned to the QoS properties

4.3. Re-calculating the QoS Scores

If a plan has been executed several times, the values of the QoS properties of the composite service will be recorded for each of the executions (i.e. instances). If several instances of a plan match the user's request, the QoS properties of the plan will be re-calculated as the weighted averages of the QoS properties of the instances. For example, if three instances of a plan match the user's request, the response time of the instances are 100ms, 120ms and 110ms respectively. The three values will be averaged and used as the response time of the plan. The reason for using the weighted averages to represent the values of the QoS properties is to ensure that the more recently executed instances have more influence to the resulting properties' values than the earlier instances. To achieve this effect, the decay function below is used to calculate the weight of each instance.

$$d(i,n) = \frac{1 - \frac{i-1}{n}}{\sum_{i=1}^{n}(1 - \frac{i-1}{n})} \quad \text{where} \quad 1 \le i \le n$$

$d(1,n)$ is the weight for the most recently executed instance, and, $d(n,n)$ is the weight for the earliest instance being executed. According to this decay function, the weights for the three instances in the above-mentioned example would be $\frac{1}{2}, \frac{1}{3}$, and, $\frac{1}{6}$. It can be seen that the most recent instance is given weight $\frac{1}{2}$ and the earliest instance is given weight $\frac{1}{6}$. Thus, the most recent instance contributes more to the resulting response time.

The value of a QoS property V for a plan can be calculated according to the plan's instances as below:

$$V = \frac{\sum_{i=1}^{n}(d(i,n) * V_i)}{\sum_{i=1}^{n} d(i,n)}$$

In the above expression, n is the total number of the instances that are used to calculate the weighted average, $d(i,n)$ is a decay function. V_i is the value of the QoS property of instance i.

The number of instances used in calculating the weighted average is given to the CSMS as a parameter. Using more instances in the calculation makes the system slow in reflecting changes to the system. However, more instances would make the weighted average more robust to noise (i.e. the abnormal fluctuation of the QoS value). Although our prototype used the above decay function, the decay function and the number of instances used in the calculation would probably have to be fine-tuned according to real applications and the operating environment.

Once the values of the QoS properties of the plans are re-computed, for each distinctive execution plan, the QoS score of the plan is computed according to formulas (1)-(3) in §2.1. When calculating the QoS score of a plan, formula (3) uses the weights assigned to the QoS properties by the current user's request. Then, the plans are passed to the selection mechanism for choosing an appropriate plan.

4.4. Selecting a Plan

The QoS score of a plan corresponds to the fitness of the plan in the roulette-wheel selection. The plans matching the user's request are sorted according to their QoS scores. The users can assign different weights to the plans of different ranking. For example, if there are two matching plans and the user prefers to use the best plan (i.e. the plan with the highest QoS score) than the second best plan by a ratio of 2 to 1, the best plan will be given a weight of 2 and the other plan will have a weight of 1. The following formula is used to calculate the probability of a plan being selected. In the formula, W_i is the weight assigned to plan i, $Score_i$ is the QoS score of plan i, and, n is the number of plans that match the current request.

$$P_i = \frac{W_i * Score_i}{\sum_{j=1}^{n}(W_j * Score_j)} \qquad (4)$$

The following rule is used for selecting a plan.

Selection Rule:

Let R be a randomly generated number such that $R \in (0,1]$

Let $\{1, \ldots, n\}$ be a collection of matching plans such that $\forall i : \{1..n\} \forall j : \{1..n\}.(i < j) \rightarrow (Score_i \le Score_j)$

Let P_k be the selection probability of plan k obtained using formula (4)

Let $k \in \{1, \ldots, n\}$ be a plan such that $\sum_{i=1}^{k-1} P_i < R \le \sum_{i=1}^{k} P_i$

Return plan k as the selected plan

For example, if there are three plans, say A, B, and, C. The QoS scores of A, B and C are 40, 30 and 10 respectively. The weights being assigned to the plans are 7, 2, and, 1. According to formula (4), the probability of A, B and C being selected is 80%, 17.1% and 2.9% respectively. Thus, according to the selection rule, if "$R \in (0, 0.8]$" holds, then plan A is selected. Plan B is selected if R is in (0.8, 0.971]. For other values of R, plan C is selected.

The operating environment determines the number of plans used in a roulette-wheel selection. If the QoS values given by the service providers are mostly inaccurate or the QoS of the service providers fluctuates

a lot, it might be a good idea to have more plans available for selection, since this would allow more plans being wrongly classified as inferior due to inaccurate information to have a chance to execute.

4.5. Generating Execution Plans

When the CSMS receives a user's request, the CSMS retrieves the instances of the plans that match the user's request from the CBR repository. Then, as described in §4.3 and §4.4, the CSMS (a) re-calculates the QoS scores for each of the distinctive plans, and, (b) executes the selection algorithm to choose a plan to execute. If no matching plan can be found in the CBR repository, as [18], the CSMS uses an IP solver to generate a plan. Before using the IP solver, the CSMS needs to contact the UDDI registry to retrieve the QoS data of all the candidate service providers for the given composite service and passes the data to the IP solver. The CSMS also needs to generate the constraints for the IP solver to use. The constraints that need to be generated are the same as the ones given in [18]. The generated plan, the QoS values observed during the execution of the plan, the weights given to the QoS properties, and, the time at which the plan is executed are stored in the CBR repository.

Since the IP solver only returns the "best" solution in terms of the QoS score, in order to provide alternative plans for the roulette-wheel selection to use, the CSMS uses the IP solver to generate alternative plans. This is carried out off-line by the CSMS. In order to make sure that the IP solver does not find the plans that already exist in the CBR repository, the following constraint needs to be given to the IP solver. In the constraint, $plan^{CS}$ is the set including all the existing plans for composite service *CS*. For a plan *p* in $plan^{CS}$, *Serv*(*p*) denotes the set that contains the IDs of all the service providers in *p*, and, |*Serv*(*p*)| denotes the number of service providers in *p*.

$$\forall p : plan^{CS} . \sum_{j \in Serv(p)} T_j < |Serv(p)|$$

where T_j is either 0 or 1

T_j equals to 0 means service provider *j* is not used in a plan. Otherwise, provider *j* is used in a plan. Thus, $\sum_{j \in Serv(p)} T_j < |Serv(p)|$ means that not all the service providers in plan *p* can be in the alternative plan.

Since new service providers might join the system and the values of the QoS properties of existing services might change, the CSMS periodically runs the IP solver for each of the composite services recorded in the CBR repository to find alternative plans for the composite services. If the alternative plans have higher QoS scores than the existing plans, the alternative plans will be recorded in the CBR repository.

5. Performance Evaluation

A prototype of the system has been implemented. In the implementation, jUDDI 0.9 rc4 is used as the UDDI registry. The database being used is MySQL. MySQL is used by both the jUDDI and the CBR repository for storing information. In order to store and retrieve information to/from the jUDDI, the UDDI4J 2.0.5 library was used. The CSMS, the UDDI registry and the CBR repository reside on different machines. Each machine runs 64 bit Linux Ubuntu 7.04 and has an Intel dual core 1.6GHz CPU with 3GB of RAM.

First, the performance of the IP solver and the CBR-based approach are compared for a composite service consisting of two parallel paths. In the first set of experiments, the composite service consists of 40 tasks with 20 tasks in each path. In the second set of experiments, there are 40 tasks in each path of the composite service (i.e. there are 80 tasks in the composite service in total).

It is assumed that (a) the execution plans for the composite service are available in the CBR repository, and, (b) the QoS properties are mostly recorded accurately. Thus, the roulette-wheel selection only considers three plans for each composite service. The CBR repository contains one million instances of 45,000 different execution plans for 15,000 different composite services. That is, there are about 22 instances for each execution plan, and, each composite service has three different execution plans. In the CBR approach, according to §4.2, the user's request can only match with the plans of one composite service in the CBR. Thus, even if all the instances of the composite service's plans match with the user's request, there are at most about 66 matching instances to be retrieved from the CBR repository. Thus, the cost for retrieving the instances does not increase as the number of tasks in a composite service increases. Two QoS properties, the response time and the cost of the service, are considered in service composition. ε_{time} and ε_{weight} are both set to large numbers to allow all instances of the plans to be matched.

The number of candidate services for each task varies from 50 to 200 with steps of 50. Figure 2 shows that the time for obtaining an execution plan from the IP solver is a lot higher than the CBR approach. This is because, in the IP solver approach, the CSMS needs to retrieve the QoS properties' values of all the candidate services from the UDDI registry as well as constructing the constraints for the IP solver. Thus, a large amount of time was spent on pulling data from

the UDDI registry. As the number of tasks in a composite service increases, the number of candidate services also increases. Hence, when there are 80 tasks in the composite service, the IP solver approach takes more time to generate an execution plan than when there are 40 tasks in the composite service. As discussed earlier, regardless of the number of tasks in a composite service, there are always about 66 instances being retrieved from the CBR repository. Thus, the response time of the CBR-based approach remains almost constant. Hence, only one set of results is plotted for the CBR-based approach.

As the figure shows, the CBR approach performs better than the IP solver approach. The CBR approach avoids spending long periods of time solving a problem that had been solved previously. In terms of the execution plan being generated/selected by the two approaches, since the IP solver generated all the plans stored in the CBR repository, the qualities of the plans obtained from the two approaches should be similar.

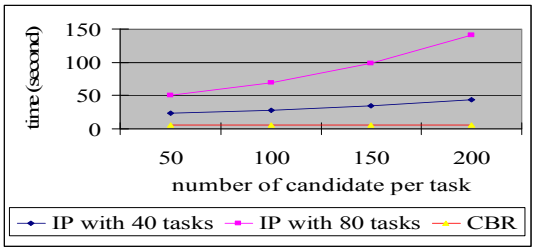
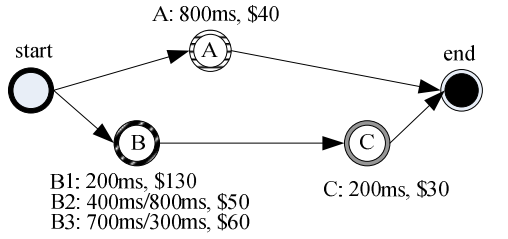


Figure 2 Comparison between IP and CBR

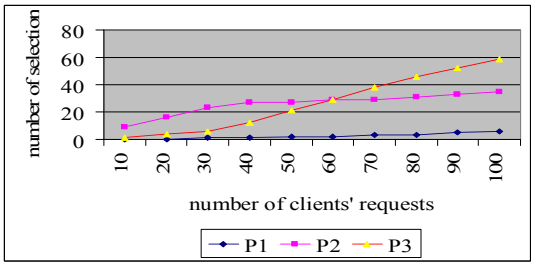
The next experiment shows the proposed scheme works well even if the information about the service providers is inaccurate. The composite service used in this experiment consists of three tasks, i.e. *A*, *B*, and, *C*, on two parallel paths as shown in Figure 3(a). There is one candidate for task *A*. The response time and the cost of the candidate are 800ms and $40 respectively. Task *C* has one candidate with a response time of 200ms and a cost of $30. There are three candidates, i.e. *B1*, *B2*, and, *B3*, for task *B*. *B1* has a response time of 200ms and a cost of $130. The cost of task *B2* is $50. The response time of *B2* fluctuates during the execution. Initially, the response time of *B2* is 400ms. After a while, *B2*'s response time deteriorates to 800ms. Candidate *B3* has a cost of $60. *B3* registers its response time as 700ms. However, *B3* can achieve a response time of 300ms during execution. Figure 3(a) shows the QoS values associated with the candidates. It is assumed that the weights being assigned to the QoS score calculation is 40% to the response time and 60% to the cost.

There are three possible execution plans, i.e. *P1*, *P2* and *P3* where *P1*={*A, B1, C*}, *P2*={*A, B2, C*}, and, *P3*={*A, B3, C*}. For a composite service with parallel paths, the service terminates when all parallel paths terminate. Thus, according to the initial response time of *B2*, i.e. 400ms, and, *B3*'s registered response time, i.e. 700ms, the maximum and the minimum response time of the three plans are 900ms and 800ms respectively. The maximum and the minimum cost of the three plans are $200 and $120 respectively. Thus, according to formulas (1) – (3) in §2.1, the QoS scores of the three plans are $Score_{P1}$ = 0.4, $Score_{P2}$ = 1, and, $Score_{P3}$ = 0.525. If the response time for plan *P3* is calculated according to the response time that *B3* can achieve (i.e. 300ms), the response time of *P3* becomes to 800ms. In this case, since the response times of the three plans are the same, their QoS scores are determined by their costs. It can be seen that *P2* is still the best plan as it has the least cost.

In the experiment, the response time of *B2* deteriorates to 800ms after the system receives 30 requests from the clients. Once the response time of *B2* becomes 800ms, the maximum and the minimum response time of the three plans become to 1000ms and 800ms. The QoS scores of the three plans become to $Score_{P1}$ = 0.4, $Score_{P2}$ = 0.6, and, $Score_{P3}$ = 0.925[1]. In this case, the best plan is *P3*. In the experiment, for the roulette-wheel selection, the weights being assigned to the best, the second best and the worst plans are 7, 2, and, 1 respectively.



(a)



(b)

Figure 3 Coping with Inaccurate QoS Information

Figure 3(b) shows the results of the experiments. It can be seen that *P2* is selected most of the time at the beginning, since *P2* is the best plan. *P1* and *P3* also

[1] $Score_{P3}$ is calculated based on the response time of *B3* is 300ms.

have some chances to be selected. The execution of *P3* helps the CBR system to learn that the response time of *P3* is actually 800ms. As the response time of *B2* becomes to 800ms after 30 requests from the client, *P3* becomes the best plan. As a result, the CSMS selects *P3* most of the time for the remaining executions.

From this experiment, it can be seen that the CSMS is able to adapt to QoS fluctuations of the service providers. Using the roulette-wheel selection helps the CSMS to identify that plan *P3* was wrongly classified. If the CSMS only uses the IP solver to choose the execution plan, *P3* will never be selected since the IP solver only returns the best plan. As a result, the system cannot discover that the response time of *B3* is actually 300ms. Consequently, *P2* will remain as the best plan even after *B2*'s response time deteriorates to 800ms. This is because $Score_{P2}$ is 0.6 when *B2*'s response time is 800ms, and, $Score_{P3}$ is 0.525 when *B3*'s response time is 700ms.

6. Conclusion and Future Work

The system proposed in this paper combines the IP solver, genetic algorithm, and, the CBR techniques to tackle the QoS-aware service composition problem. The system reduces the amount of time spent on solving a service composition problem by reusing previous execution plans. The experiment shows that, when there are sufficient amount of execution plans in the CBR repository, the proposed system can outperform the IP solver approach in finding an execution plan for complex composite services. The experiment also shows that the proposed scheme adapts well to the changes of the service qualities of the service providers. In addition, it shows that, unlike the solutions based purely on the IP solver, the system works well when the information about the service providers is inaccurate.

The values of the QoS properties are important in deciding the selection of the service providers. The CSMS plays the role of the execution engine. Thus, it can measure the QoS values of the services in an unbiased way. If the clients prefer to conduct the executions themselves, the clients will be responsible for monitoring the QoS properties. Since the clients and the service providers might have incentives for not reporting the QoS values accurately, it is important to have a mechanism for measuring the QoS properties reliably. Many research have been carried out to ensure that QoS properties can be monitored accurately, e.g. [13], [8], etc. We will investigate how to decentralize the execution engine to allow the clients to conduct the execution of their composite services while ensuring the QoS properties are recorded accurately.

References

[1] R. Cheng, S. Su, F. Yang, and Y. Li, Using Case-Based Reasoning to Support Web Service Composition, 6th Intl. Conf. on Computational Science, LNCS 3994, pp. 87 – 94, 2006.

[2] J. Day and R. Deters. Selecting the best web service, Proc. of the 2004 conference of the Centre for Advanced Studies on Collaborative research, pp. 293–307, 2004.

[3] Ernest J. Friedman-Hill. Jess, the java expert system shell. May 2001. http://herzberg.ca.sandia.gov/jess

[4] R. Howard and L. Kerschberg. A framework for dynamic semantic web services management. Intl. Journal of Cooperative Information Systems, 13(4):441–485, 2004.

[5] R. Howard and L. Kerschberg. A knowledge-based framework for dynamic semantic web services brokering and management, 15th International Workshop on Database and Expert Systems Applications, pp. 174–178, 2004.

[6] C.-L. Hwang and K. Yoon. Multiple Criteria Decision Making. Lecture Notes in Economics and Mathematical Systems. Springer, 1981.

[7] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl. QoS aggregation for service composition using workflow patterns, Proc. of the 8th Intl. Enterprise Distributed Object Computing Conference, pp. 149–159, 2004

[8] R. Jurca, B. Faltings, and W. Binder. Reliable qos monitoring based on client feedback, Proc. of the 16th Intl. conference on World Wide Web, pp. 1003–1012, 2007

[9] J. L. Kolodner. Case-based reasoning. Morgan Kaufman, 1993

[10] S. Lajmi, C. Ghedira, K. Ghedira, and D. Benslimane. Wesco cbr: How to compose web services via case based reasoning. 2006 IEEE International Conference on e-Business Engineering, pp. 618–622, 2006.

[11] Y. Li , J. Huai , T. Deng, H. Sun , H. Guo , Z. Du, QoS-aware Service Composition in Service Overlay Networks, 2007 IEEE Int'l Conf. on Web Services, pp. 703-710, 2007

[12] B. Limthanmaphon and Y. Zhang. Web service composition with case-based reasoning. Proc. of the 14th Australasian database conference, pp. 201–208, 2003

[13] Y. Liu, A. H. Ngu, and L.Z. Zeng. Qos computation and policing in dynamic web service selection, 13th international World Wide Web conference, pp. 66–73, 2004

[14] C. R. Reeves and J. E. Rowe. Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory. Kluwer Academic Publishers, 2002

[15] D. Thiβn and P. Wesnarat. Considering qos aspects in web service composition. 11th IEEE Symposium on Computers and Communications, pp.371–377, 2006.

[16] N. Thio and S. Karunasekera. Automatic measurement of a qos metric for web service recommendation, Proceedings of Australian Software Engineering Conference, pp. 202–211, 2005.

[17] T. Yu and K. J. Lin, "Service Selection Algorithms for Web Services with End-to-end QoS Constraints", Journal of Information Systems and e-Business Management, 2005.

[18] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. IEEE Trans. Softw. Eng., 30(5):311–327, 2004.