Journal of Web Engineering, Vol. 13, No. 1&2 (2014) 053–066

PROCESSING MULTIPLE REQUESTS TO CONSTRUCT SKYLINE COMPOSITE SERVICES

SHITING WEN
Ningbo Institute of Technology, Zhejiang University, Ningbo, China
wst1029@mail.ustc.edu.cn

QING LI
City University of Hong Kong, Hong Kong, China
itqli@cityu.edu.hk

CHAOGANG TANG
China University of Mining and Technology, Xuzhou, China
tcg@mail.ustc.edu.cn

AN LIU[a] LIUSHENG HUANG
University of Science and Technology of China, Hefei, China
liuan@ustc.edu *lshuang@ustc.edu.cn*

YANGGUANG LIU
Ninbo Institute of Technology, Zhejiang University, Ningbo, China
ygliu@nit.zju.edu.cn

Received April 1, 2013
Revised July 25, 2013

The performance of a composite service is determined by the performance of involved component services. When multiple non-functional criteria are considered, users are required to express their preferences over different quality attributes as numeric weights in existing methods. However, this imprecise method may not reflect the natural ordering of services and thus could miss some user-desired services. In this paper, we propose a composition framework to construct multiple skyline composite services for each individual request. We also discuss how a service registry can effectively deal with multiple requests simultaneously by materializing the intermediate composite services. We evaluate the efficiency and effectiveness of our methods through extensive experiments.

Keywords: Web Services; Skyline Composition; Materialization
Communicated by: D. Schwabe & E.-P. Lim

1 Introduction

Web service composition has been an emerging methodology for building modern business applications [1]. The functional granularity of services should be constrained from a reusability point of view [2]. Thus, multiple services need to be composed into a new value-added one, resulting in a composite service, to fulfill users' complex requirements. Specifically, a

[a]Corresponding Author

composite service is specified as an abstract process composed by a set of abstract tasks. Prior to the run time, it has to bind a concrete service for each abstract task. QoS-based service composition aims at finding the best combination of web services that satisfy a set of end-to-end QoS constraints with maximum utility, which is defined by users according to their preferences. To the best of our knowledge, all utility functions use weight-additive methods to obtain a single numerical value and compare this value to select a service to invoke (e.g., [3, 4, 5, 6, 7, 8]). This is an optimization problem known as NP-hard. In particular, how to determine the weight of each dimension of QoS is non-trivial, and the composer can only return one composing result for each request according to the utility value. The fairness of these methods is determined by the weight of each QoS dimension, but such a weight is hard to be assigned objectively. Indeed, ensuring the fairness by means of simple weigh-additive methods is almost impossible in service composition.

Recognizing the shortcomings of traditional composition methods, there are has been quite some research on constructing composite services through the skyline technique [9] which, unlike traditional methods, does not assign any weight to each QoS dimension in the process of composition (e.g., [10, 11, 12]). According to such skyline-based composition, only those services belonging to the skyline (i.e., those not dominated by any other functionality-equivalent services), are valid candidates for the composition. This can provide an initial pruning of candidate services. However, performing an exhaustive search by finding all skyline composition results is time consuming. We observe that skyline-based composition may produce many intermediate composition results. Furthermore, multiple requests can share those intermediate results, similar to the case of multi-query processing in data warehouses [13]; indeed, multiple requests may overlap on some common intermediate composition results, and common sub-composition results may appear multiple times for a complex request. In this paper, we consider materializing some intermediate skyline composition results for accelerating composition since the same request may be issued multiple times by different users. The concept of materializing indicates to store the results (intermediate results) of past requests for reuse to answer a new request. Our main work and contributions can be summarized as follows:

- We propose a formal model for dealing with multiple service composition requests, through materializing a set of intermediate composite service classes. Our model can return a set of skyline candidate solutions for each request.

- We address the problem of materializing a set of intermediate skyline composite service classes by adopting the Mix Integer Programming (MIP) method.

- We also propose a Heuristic Materializing Approach (HMA) to select a set of appropriate intermediate composite service classes to materialize, so as to reduce the complexity of the MIP method.

The rest of the paper is organized as follows. Section 2 presents a motivating example and gives our problem statement. Section 3 presents a framework to constructing skyline composite services. Section 4 presents our algorithms for processing multiple requests. Section 5 evaluates the performance of our proposed algorithms. Section 6 discusses some related works. Finally, Section 7 concludes the paper and sheds light on future research.

Table 1. Computation Cost for Calculating Skyline Solutions

Service Class	Non-Materialized	Materialized
Transports	337.5	12.5
Visit Friend	2290	31.25
Travel Process	13038	78.125
Attend Conf.	318995	390.25

2 Motivating Example and Problem Statements

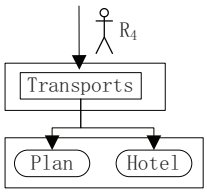
2.1 Motivating Example

In order to improve reusability of a service, a provider may offer different granularities for different users' requests. Figure 1(e) shows an example of a global composition plan, where the square nodes indicate composite service classes, and oval nodes indicate atomic service classes. Particularly, the oval nodes and square nodes provide, respectively, services at atomic granularity and composite granularity for users' invocation.

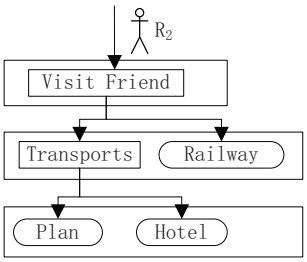
Suppose that for illustrative purpose each atomic service class has 10 services and half of them are skyline services. We first show the benefits of materializing a set intermediate composite service classes to accelerate processing of a composition request in the service registry. We consider four frequent composition requests: r_1: transports, r_2: visiting a friend, r_3: travel process, and r_4: attending a conference.

When a service provider receives a request, it may select a suitable level of invocation in the global service composition plan (c.f. Figure 1(e)). Figure 1((a)-(d)), the provider generates four sub-composition plans cp_1, cp_2, cp_3 and cp_4 for requests r_1, r_2, r_3, and r_4, respectively. In order to achieve fast request processing, some intermediate composite service classes can be materialized because different users may send their requests multiple times. To show the difference of the cost on obtaining all skyline solutions with and without materializing the intermediate composite service classes, we use request r_1 as an example. If we do not materialize any intermediate nodes, the cost of replying all skyline candidate solutions for r_1 includes two parts: 1) the cost of calculating all candidate composite solutions (cost is 25, namely, travel times); 2) the cost on filtering the skyline candidate solutions (cost is 252/2=312.5). However, if we materialize the intermediate composite service class Transport, the cost for obtaining all of skyline solutions is 12.5. For simplicity, we assume that the approach for constructing skyline composite service classes uses linear search and nested loop techniques. Therefore, the cost for request processing and maintaining intermediate composite service classes are determined by the size of the involved service classes. Table 1 lists the cost for calculating all of the skyline solutions for requests r_1, r_2, r_3, and r_4, respectively.

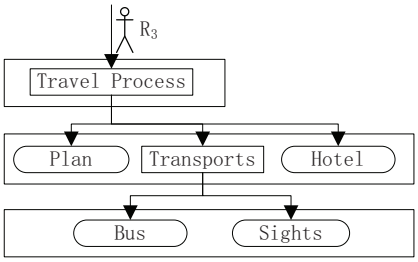
While materializing some intermediate composite service classes can save the average request processing time, it still produces an additional maintenance cost for updating those intermediate composite service classes if their atomic service classes are changed. In fact, how to balance the additional maintenance cost with the saving from materialization is an optimization problem [13]. In the next sub-section, we give a formalized definition of our problem.



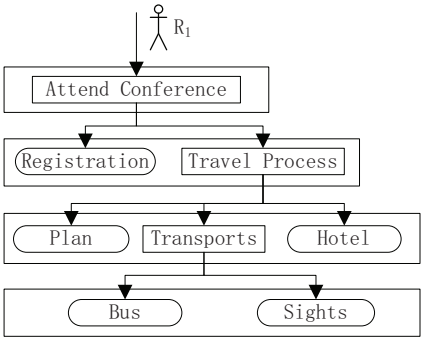
(a) Composition Plan for r_1



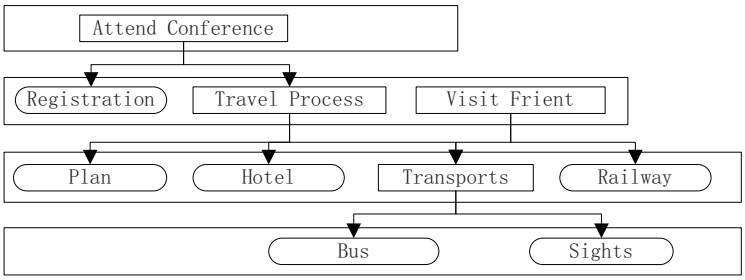
(b) Composition Plan for r_2



(c) Composition Plan for r_3



(d) Composition Plan for r_4



(e) A Global Composition Plan

Fig. 1. Composition Plan of Each Request

2.2 Problem Statement

We formalize the problem of efficiently processing multiple composition requests for composing all skyline composite services as follows:

Definition 1 *(*Optimal Selection*) Let $I = \{1, 2, ..., n\}$ be a set of intermediate composite service classes. Each element $i \in I$ binds to f_i^r which is an access frequency derived from users' requests, and f_i^s which is an update frequency derived from updated atomic service classes'. The optimal selection is to find a set of intermediate composite service classes to materialize, so as to minimize the total cost of processing all requests subject to the maximal space available for materialization.*

3 Constructing Skyline Composite Services

In our model, we use $\mathcal{S}$ to denote a set of service classes which classify the universe of available web services according to their functionality. More specifically, $S_i = \{s_{i,1}, s_{i,2}, ..., s_{i,n}\}$ a services class, $S_i \in \mathcal{S}$ represents a set of functionality-equivalent web services (e.g., flight booking services) with different non-functional properties (e.g., at different response time and different prices).

Given a set of points in a d-dimensional space, a skyline query selects those points that are not dominated by any other points [9]. A point x is said to dominate another point y, if x is better than or equal to y in all dimensions and strictly better in at least one dimension.

Definition 2 *(*Dominance*) Consider a service class S, and two services $x, y(\in S)$ characterized by a set Q of QoS attributes. We say x dominates y, denoted as $x \succ y$, iff x is as good or better than y in all parameters in Q and better in at least one parameter in Q, i.e., $\forall k \in [1, |Q|] : q_k(x) \leq q_k(y)$ and $\exists k \in [1, |Q|] : q_k(x) < q_k(y)$.*

Definition 3 *(*Skyline Services*) The skyline services of a service class S, denoted by SLs, comprise those services in S that are not dominated by any other services, i.e.,$SLs = \{x \in S | \neg \exists y \in S : y \prec x\}$.*

Definition 4 *(Skyline Composition) Given a set of component services and a user request r, we compose all candidate solutions such solutions are termed as skyline compositions in which the overall QoS is not dominated by each other.*

A straightforward method for finding all skyline composite services is by iterating and comparing all possible combinations of candidate services, which is very time-consuming. In the following section, we address this problem by materializing some intermediate composite service classes, with an aim to improve the performance of processing multiple user requests of constructing all skyline composite services as candidate solutions.

Figure 2 gives an overview of our skyline-based composition framework which involves four types of entities: Service, Service Registry, Customer, and Composer. The service publishes its basic information (e.g., input parameters, output parameters, and QoS parameters) in the Service Registry when it is ready for use (Step (1)). Composition begins when a user sends to the Composer a request that contains the required functionality and personal preferences (Step (2)). After receiving the request, the Composer makes use of the search ability of the Service Registry to find out all the possible candidate skyline composite services that can fulfill the user's requirements. A composition will be activated if no single service can fulfill the user's requirements. In this case, several services will be selected and composed

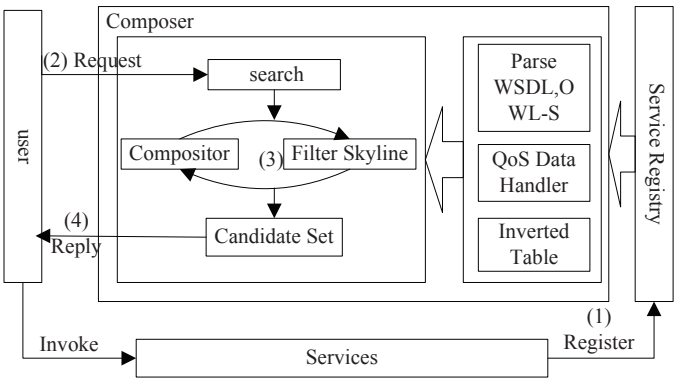


Fig. 2. System Architecture

as a temporary composite service and recommended to the user if the composite service is not dominated by any other composite service (Step (3)). The user selects and invokes a composite service from the set of candidate composite services (Step (4)) recommended to him/her.

In our approach, we first prune all non-skyline sub-services which have the same functionality non-skyline in order to keep the sub-service candidates as small as possible. By focusing only on the skyline services with respect to each service functionality, we can speed up the composition process while still being able to find all candidate skyline solutions.

4 Multiple Request Processing

4.1 Global Composition Plan

As shown in Figure 2, the Composer can use the search ability of the service Registry to obtain, based on a batch of composition requests, a global composition plan like the one in Figure 1(e).

Our goal is to find a set of intermediate composite service classes to materialize, so that the total cost is minimum. In this regard, we make several observations as follows: 1) to achieve the best performance, an intuitive way is to materialize all the intermediate composite service classes, which, however, will incur the highest cost of maintenance; 2) to achieve the lowest maintenance cost, the easiest way is to leave all intermediate composite service classes virtual, which, however, will result in the poorest performance; 3) if we choose to materialize some of the intermediate composite service classes, and leave some of them virtual, we may achieve an optimal composition performance in terms of the total cost of processing requests and maintaining materialized intermediate composition results.

To this end, we define a global composition plan (GCP) as a directed acyclic graph that represents a processing plan of multiple requests. In particular a GCP is a labeled DAG $G = (V, A, C_R, C_U, F_R, F_S)$, where V is a set of nodes in GCP and A is a set of arcs over V. Here, the vertex set V includes three types of vertexes (i.e., user, atomic service class, and intermediate service class), and we use $r \in V$ to represent a user request, use $s \in V$ to represent a basic service class, and $i \in V$ to represent an intermediate service

class, respectively. Further, $c_i^p \in C_R$ represents the cost to access an intermediate service class i, and $c_i^u \in C_U$ indicates the cost to update an intermediate composite service class i. Lastly, $f_i^r \in F_R$ represents the accessing frequency of intermediate service class i, and $f_i^s \in F_S$ represents the update frequency of intermediate service class i, respectively. We distinguish the frequencies of user requests from the frequencies of accessing intermediate composite service classes. Suppose we have a set of requests $R = \{r_1, r_2, ..., r_{|R|}\}$ and each request r_j has a frequency f_{r_j}. We have the following formula:

$$f_i^r = \sum_{j=1}^{|R|} f_{r_j} \times x_{r_j,i} \tag{1}$$

where $x_{r_j,i}$ is set to 1 if the request r_j needs to access the intermediate service class i, and 0 otherwise. Similarly, we distinguish update frequencies of intermediate composite service classes from the basic service classes' update frequencies. Note that the update of the intermediate composite service classes is derived from the update of basic service classes. Suppose, there is a set of basic service classes $\mathcal{S} = \{S_1, S_2, ..., S_{|\mathcal{S}|}\}$, and each basic service class S_j has an update frequency f_{S_j}. We have the following formula:

$$f_i^s = \sum_{j=1}^{|S|} f_{S_j} \times x_{S_j,i} \tag{2}$$

where $x_{S_j,i}$ is set to 1 if the intermediate composite service class i derives from basic service class S_j, and 0 otherwise.

4.2 Using Mix Integer Programming

How to select the set M of intermediate service classes to materialize depends on four factors: 1) the accessing frequency of each intermediate composite service class; 2) the updating frequency of each intermediate composite service class; 3) the cost of accessing each intermediate service class; 4) the cost of maintaining each intermediate composite service class. Thus, we can calculate the request processing cost and maintenance cost through formulas (3) and (4).

In particular, the cost of accessing the intermediate composite service class i is as follows:

$$Cost_P = f_i^r \times c_i^p \tag{3}$$

The cost of maintaining the intermediate composite service class i is given below:

$$Cost_M = f_i^s \times c_i^u \tag{4}$$

Then, we can calculate the total cost of processing all requests as follows:

$$\begin{aligned} Cost_{total}(M) \quad &= \textstyle\sum_{i \in M} Cost_P + \sum_{i \notin M} Cost_U \\ &= \textstyle\sum_{i \in M} f_i^r \times c_i^p + \sum_{i \notin M} f_i^s \times c_i^u \end{aligned} \tag{5}$$

Given a *GCP*, Our task is to find a set of materialized intermediate composite service classes such that the total cost of request processing and intermediate results maintenance is the minimal. Here, we advocate using a mix integer programming (*MIP*) method to solve this problem directly. There are three kinds of input in *MIP*, i.e. variables, objective function,

and constraints on the variables, in which both the objective function and constraints must be linear. *MIP* attempts to maximize (minimize) the value of the objective function by adjusting the values of variables based on the constraints. The output of *MIP* is the maximum (minimum) value of the objective function as well as the values of variables at this maximum (minimum) point.

The objective function is in particular based on formula (3) and (4). Mathematically, it can be expressed as:

$$\begin{array}{cc} Min & \sum_{i=1}^{n} f_i^r \times c_i^p \times (1 - x_i) + \sum_{i=1}^{n} \times c_i^u \times x_i \\ Subject\ to & \sum_{i=1}^{n} space(i) \times x_i \leq space_{avail} \\ & x_i \in \{0, 1\} (1 \leq i \leq n = |I|) \end{array} \tag{6}$$

Here, the variable x_i is set to 1 if the intermediate composite service class i is selected, and 0 otherwise. Further, $space_{avail}$ is the available space for materializing the intermediate composition results. Note that we assume here all basic service classes have been materialized. The function $space(i)$ can calculate the size of space occupied by the intermediate service class i. If we adopt the standard MIP solver for the global optimization problem, the time complexity is $O(2^n)$. In the next section, we propose a heuristic algorithm for this problem to aleviate the expense.

Algorithm 1: Initial Materializing Composition Results (IMCR)

input : GCP, $space_{avail}$
output: M:*a Set of Composite Service Classes*
begin
  if $space_{avail} < \sum_{S \in SS} space(S)$ **then**
    $return: \ M \leftarrow \emptyset$;
  else
    foreach $S \in SS$ **do**
      $M \leftarrow M \cup \{S\}$;
      $space_{avail} \leftarrow space_{avail} - space(S)$;
    end
  end
  $sList \leftarrow Sort\ I\ according\ to\ gain(i): \ i \in I$;
  while $sList \neq \emptyset$ **do**
    $i \leftarrow sList$;
    if $space(i) > space_{avail}$ **then**
      $remove: \ sList \leftarrow sList/\{i\}$;
    else
      $M \leftarrow M \cup \{i\}$;
      $space_{avail} \leftarrow space_{avail} - space(S)$;
      $remove: \ sList \leftarrow sList/\{i\}$;
    end
  end
  $return: \ M$;
end

4.3 Heuristic Algorithm

Our heuristic algorithm consists of two parts: 1) initialization of materializing composite service classes (*IMCR*); and 2) update of materializing composite service classes (*UMCR*). The aim of *IMCR* is to materialize the intermediate composite service classes according to users' previous request frequencies, basic service classes' update frequencies, and available space for materialization. The *UMCR* algorithm is to deal with the situation where new intermediate composite service classes need to be materialized. Moreover, how to replace an intermediate composite service class from the set of materialized composite service classes depends on the gain and loss metrics if the available space does not suffice to materialize the new composite service class. Below, we first introduce some notations to be used in our heuristic algorithm.

- $Cost_{total}(M)$ denotes the total cost for processing all users' requests with materializing a set of intermediate composite service classes. M represents the set of materialized intermediate service classes.

- $gain(i)$ denotes how much gain we will earn from materializing an intermediate composite service class i. Using this gain metric, we can assess the cost-effectiveness of each intermediate composite service class and select a class with a higher gain metric as a candidate for materialization. The definition of the gain metric is given below:

$$gain(i) = \frac{Cost_{total}(M) - Cost_{total}(M \cup \{i\})}{space(i)} \tag{7}$$

  As shown in formula 7, we compute the difference between the total cost with M and the total cost after the addition of an intermediate service class i, and divide the cost difference by the space required for materializing the additional intermediate composite service class i. In our algorithm, we prefer to select an intermediate composite service class with a higher gain metric for materialization in order to achieve better performance in cost reduction. However, an intermediate composite service class with a high gain metric will not be selected if its occupied space exceeds the actual available space.

- $loss(i)$ denotes how much loss will incur by removing an intermediate composite service class i from the materialized set M. Using this loss metric, we can assess the cost-effectiveness of each intermediate composite service class, and select a class with a lower loss metric as a candidate for deletion. The definition of the loss metric is shown as follows:

$$loss(i) = \frac{Cost_{total}(M/\{i\}) - Cost_{total}(M)}{space(i)} \tag{8}$$

  As shown, we compute the difference between the total cost with M and the total cost after the deletion of an intermediate service class i, and divide the cost difference by the space released after removing from M the intermediate composite service class i. In our algorithm, we prefer to select an intermediate composite service class with a lower loss metric for removing in order to achieve better performance in terms of cost reduction.

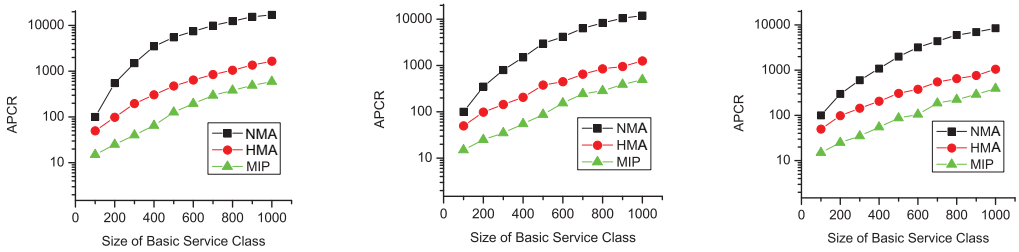
Algorithm 1—the *IMCR* algorithm is to select the intermediate composite service classes with the largest value of *gain metric*(c.f. Lines 12-21). The aim is to materialize the basic service classes first(Lines 02-09) if the available space is enough. It sorts all intermediate service classes according to their values of the gain metric (Line 10). In addition, Algorithm 2—the *UMCR* algorithm is to add new intermediate composite service classes to the materialized set if there is available space (Lines 02-06). Otherwise, it will delete one or more materialized composite service classes with the lower loss metric values, and materialize a new intermediate composite service class whose gain metric is higher than that of the replaced intermediate composite service classes (Lines 07-23). Interestingly, we can see that both Algorithms 1 and 2 have the $O(n)$ time complexity, i.e., they are of linear time complexity.

Algorithm 2: Updating Materialized Composition Results(UMCR)

```
   input  : M, space_avail, i', I
   output: M:a Set of Composite Service Classes
 1 begin
 2   if space_avail ≥ space(i') then
 3     M ← M ∪ {i'};
 4     space_avail ← space_avail − space(i');
 5     return : M;
 6   end
 7   i ← min(loss{I});
 9   while I ≠ ∅ do
10     tempSet ← tempSet ∪ {i};
11     if gain(i') > gain(tempSet) then
12       if space_avail + Σ_{i∈tempSet} space(i) ≥ space(i') then
13         M ← M/{tempSet};
14         M ← M ∪ {i'};
15         space_avail = space_avail + Σ_{i∈tempSet} space(tempSet) − space(i');
16         return : M;
17       else
18         I ← I/{i};
19         i ← min(loss{I});
20       end
21     end
22   end
23   return : M;
24 end
```

5 Experiments

In this section, we evaluate the effectiveness of our proposed method by conducting extensive experiments.



(a) Anti-Correlated (b) Independent (c) Correlated

Fig. 3. Average Processing Cost per Requests (APCR) vs. Size of Basic Service Class

5.1 Simulation Setup

Given a service class, the size of the skyline services depends on the distribution of the QoS values and correlations between different QoS parameters. These include: 1) the correlated dataset, in which a service that is good in one dimension is also good in another dimension; 2) the independent dataset, in which the values of two QoS dimensions are independent of each other; and 3) the anti-correlated dataset, in which there is a clear trade-off between a pair of dimensions. Note that the number of skyline services is relatively small in the correlated dataset, large in the anti-correlated, and medium in the independent ones.

All experiments are performed on a PC with 2.2GHz Intel Pentium Duo2 CPU, 2048M of RAM. Microsoft Windows 7 Operating Systems, J2SDK 1.6. To make the experimental result comparable, our experiments are conducted on a publicly available dataset EEE05 [14]. However, this dataset only has the WSDL files of services, and the QoS information is not contained. In order to test our approach with a larger number of services and different distributions, we use each WSDL file of EEE05 as a service class and use a publicly available synthetic generator [15] to obtain three different datasets. Each service's QoS is represented by a vector of four dimensions, namely, response time, availability, price, and reputation.

We evaluate the following three methods: non-materialized method as baseline method (NMA), mix integer programming method (MIP), and our proposed heuristic materialization approach (HMA), respectively. We apply these methods to a set of composite service plans generated by test requests of EEE05 dataset, then merge all these plans to generate a single global composition plan.

5.2 Average Processing Cost

By inspecting carefully the intermediate composite service classes which are selected for materialization, we find that the efficiency of our algorithm may be affected by the size of basic service classes. So we evaluate below how the size of the basic service classes impacts on the average processing cost per request ($APCR$), by varying the size of basic service class from 100 to 1000. For an intermediate composite service class i, we set the access frequency and updat frequency $f_i^r = 50$ and $f_i^s = 10$ times per time unit, respectively. The available space for materializing intermediate composition results is assumed to be of 50%. Figure 3 shows the average processing cost when different types QoS information are applied. The X-axis represents the size of the basic service classes, and the Y-axis indicates the $APRC$. We

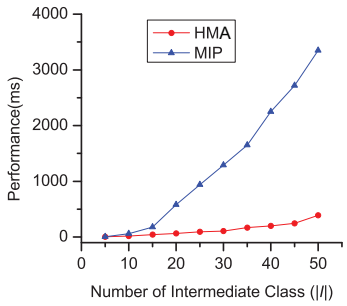


Fig. 4. Performance Evaluation

can observe that the non-materialized approach (NMA) has larger average process cost than that of the MIP method and our heuristic materialized method for all types of the datasets. Moreover, while the $APCR$ of the heuristic method is larger than that of the MIP, the gap is reasonably narrow on all the datasets. More specifically, no matter which type of the dataset the is used, the average request processing cost of the heuristic method is always close to the average request processing cost of the MIP method. Lastly, as Figure 3 shows, the average processing cost generally goes up when the size of the basic service classes increases with the anti-correlated dataset incurring the largest $ACPR$, the independent data set incurring the midum $ACPR$, and the correlated dataset incurring the lowest $ACPR$. In addition, we also notice that there is an exponential increase in the method of NMA. However, by increasing the number of basic services of each class, the required $APCR$ by both MIP and HMA methods for all the three datasets increase slowly when compared to the NMA method.

5.3 Performance Evaluation

In this part of experiments, we investigate the performance of the heuristic materialization approach (HMA) and mix integer programming (MIP) method by varying the number of intermediate composite service classes. Since the non-materialize approach (NMA) does not materialize any intermediate composite service class, it has no overhead on storing the intermediate composite service classes hence is not comparable. To compare the performance of HMA and MIP, we vary the number of intermediate composite service classes from 5 to 50. Runtime overheads mainly involve the cost of deciding which intermediate composite service class should be materialized and to which extent the cost can be saved by materializing this service class. From Figure 4, we can see that the HMA approach is indeed much more efficient than MIP, consistent with the fact that the complexity of HMA is $O(n)$ but the complexity of MIP is $O(2^n)$.

6 Related Work

Web services composition has been studied extensively in recent years. Generally, there are two types of composition: *manual* and *automatic*. The former manually defines a process consisting of multiple tasks, and its objective is to bind these tasks to concrete services while satisfying users' QoS constraints. Hence, it is also called QoS-aware service composition. In [3], the authors consider it as an integer programming (IP) problem in which the objective

function is defined as a linear composition of multiple QoS parameters. In [7], the authors also adopt IP but they use a different method to eliminate loop constructs in the process of composite services. Yu et al. [4] propose a combinatorial model and a graph model. The combinatorial model considers service composition as a multi-dimensioned multiple choice of 0-1 knapsack problems. The graph model considers service composition as a multi-constraint optimal path problem. They propose a polynomial heuristic algorithm for the combinatorial model and an exponential heuristic algorithm for the graph model. Alrifai and Riss [5] decompose global QoS constraints into an optimal set of local QoS constraints by using IP technique. The satisfaction of local constraints guarantees the satisfaction of global constraints. Through the decomposition of global constraints, it is only necessary to conduct several local selections simultaneously, which significantly improves the performance of the composition process.

Automatic service composition also dynamically generates composite services, and current approaches for automatic service composition are mainly based on AI planning [16]. In [17], the authors propose a planning algorithm called *WSPR*, whichi is essentially in accordance to the graph plan of [18], but differs in that it adopts a heuristic to minimize the number of services in a solution while the graph plan aims at minimizing the number of steps but keeping necessary actions. However, *WSPR* ignores QoS constraints, so a composite service sometimes cannot fulfill the user QoS requirements. While the work of [19] has considered QoS and counted QoS as only one-dimensioned vector, the method presented in [19] is hard to be extended to multiple QoS dimensions. Our work not only dynamically generates a composite service, but also search all skyline candidate solutions for a given user request.

7 Conclusions

In this paper, we have proposed a materialized approach to accelerate service composition. Our method achieves a low cost on request processing, which is guaranteed by two conditions. One is that multiple requests have the ability to share some sub-composite services. The other is that the frequency of updating atomic service classes is typically lower than the frequency of arriving requests. To find which intermediate composite service classes should be materialized, we have proposed a MIP and a heuristic algorithm (HMA) algorithm with some experimented studies.

In subsequent research, we will further investigate the effect of limited space on materializing intermediate composite service classes. In addition, we will consider whether grouping users' request can reduce requests' processing cost.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. 61003044 and 60873234, and Zhejiang Provincial Natural Science Foundation of China under Grant No. Y1101202.

References

1. A. Liu, Q. Li, L. Huang, and M. Xiao, "Facts: A framework for fault-tolerant composition of transactional web services," *Services Computing, IEEE Transactions on*, vol. 3, no. 1, pp. 46–59, 2010.
2. R. Haesen, M. Snoeck, W. Lemahieu, and S. Poelmans, "On the definition of service granularity and its architectural impact," in *Advanced Information Systems Engineering*. Springer, 2008, pp.

375–389.
3. L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.
4. T. Yu, Y. Zhang, and K. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, p. 6, 2007.
5. M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 881–890.
6. M. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui, "A qos broker based architecture for efficient web services selection," in *Proceedings 2005 IEEE International Conference on Web Services ICWS 2005*. IEEE, 2005, pp. 113–120.
7. D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, 2007.
8. S. Wen, Q. Li, L. Yue, A. Liu, C. Tang, and F. Zhong, "Crp: context-based reputation propagation in services composition," *Service Oriented Computing and Applications*, vol. 6, no. 3, pp. 231–248, 2012.
9. S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of the 17th International Conference on Data Engineering*. IEEE, 2001, pp. 421–430.
10. Q. Yu and A. Bouguettaya, "Computing service skyline from uncertain qows," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 16–29, 2010.
11. M. Alrifai, D. Skoutas, and T. Risse, "Selecting skyline services for qos-based web service composition," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 11–20.
12. D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. Sellis, "Top-k dominant web services under multi-criteria matching," in *Proceedings of the 12th international conference on extending database technology: advances in database technology*. ACM, 2009, pp. 898–909.
13. J. Yang, K. Karlapalem, and Q. Li, "Algorithms for materialized view design in data warehousing environment," in *Proceedings of the International Conference on Very Large Data Bases*. Institute of Electrical and Electronics Engineering(IEEE), 1997, pp. 136–145.
14. M. Blake, K. Tsui, and A. Wombacher, "The eee-05 challenge: A new web service discovery and composition competition," in *Proceedings of IEEE International Conference on e-Technology, e-Commerce and e-Service, EEE 2005*. IEEE, 2005, pp. 780–783. [Online]. Available: http://ws-challenge.georgetown.edu/ws-challenge/The%20EEE.html
15. "Random data generator:http://randdataset.projects.postgresql.org/."
16. J. Rao and X. Su, "A survey of automated web service composition methods," *Semantic Web Services and Web Process Composition*, pp. 43–54, 2005.
17. S. Oh, D. Lee, and S. Kumara, "Web service planner (wspr): An effective and scalable web service composition algorithm," *International Journal of Web Services Research (IJWSR)*, vol. 4, no. 1, pp. 1–22, 2007.
18. A. Blum and M. Furst, "Fast planning through planning graph analysis," *Artificial intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.
19. W. Jiang, C. Zhang, Z. Huang, M. Chen, S. Hu, and Z. Liu, "Qsynth: a tool for qos-aware automatic service composition," in *2010 IEEE International Conference on Web Services (ICWS)*. IEEE, 2010, pp. 42–49.