Selecting Skyline Services for QoS-based Web Service Composition

Mohammad Alrifai
L3S Research Center
University of Hannover
Germany
alrifai@L3S.de

Dimitrios Skoutas
L3S Research Center
University of Hannover
Germany
skoutas@L3S.de

Thomas Risse
L3S Research Center
University of Hannover
Germany
risse@L3S.de

ABSTRACT

Web service composition enables seamless and dynamic integration of business applications on the web. The performance of the composed application is determined by the performance of the involved web services. Therefore, non-functional, quality of service (QoS) aspects (e.g. response time, availability, etc.) are crucial for selecting the web services to take part in the composition. The problem of identifying the best candidate web services from a set of functionally-equivalent services is a multi-criteria decision making problem. The selected services should optimize the overall QoS of the composed application, while satisfying all the constraints specified by the client on individual QoS parameters. In this paper, we propose an approach based on the notion of skyline to effectively and efficiently select services for composition, reducing the number of candidate services to be considered. We also discuss how a provider can improve its service to become more competitive and increase its potential of being included in composite applications. We evaluate our approach experimentally using both real and synthetically generated datasets.

Categories and Subject Descriptors

H.3.5 [**On-line Information Services**]: Web-based services; H.3.4 [**Systems and Software**]: Distributed systems

General Terms

Management, Performance, Measurement

Keywords

Web Services, QoS, Optimization, Service Composition

1. INTRODUCTION

Recently, there has been a growing trend for businesses to outsource parts of their processes, so as to focus more on their core activities. In addition, Web users often need to compose different services to achieve a more complex task that cannot be fulfilled by an individual service. Web services provide the means for such seamless integration of business processes across organizational boundaries. Industry standards, namely WSDL, UDDI, WS-BPEL, exist for describing, locating and composing web services.

Following the Service-oriented Architecture paradigm, composite applications are specified as abstract processes composed of a set of abstract services. Then, at run time, for each abstract service, a concrete web service is selected and used. This ensures loose coupling and flexibility of the design. *Quality of Service (QoS)* parameters (e.g. responsiveness, availability, throughput) play a major role in determining the success or failure of the composed application. Therefore, a *Service Level Agreement (SLA)* is often used as a contractual basis between service consumers and service providers on the expected QoS level. QoS-based service composition aims at finding the best combination of web services that satisfy a set of end-to-end QoS constraints in order to fulfill a given SLA.

Example. Figure 1 shows an example of a web application for finding the best used car offers. The users submit their requests to the system, specifying some criteria for selecting the cars (e.g. brand, type, model). The system then returns a list of the best offers along with a credit and an insurance offer for each car on the list. The composed application can be exposed to users as a web service, API or widget, programmatically accessible or directly integrated into their web applications using a Mashup tool.

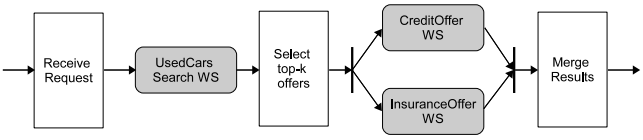

Figure 1: Example of Service Composition

In this example, some tasks like *UsedCarSearch* or *CreditOffer* are outsourced (illustrated as gray boxes in Figure 1) and integrated via web service calls. For these outsourced tasks, multiple services may be available providing the required functionality but with different QoS values. Users are typically unaware of the involved services, and they specify their QoS requirements in the SLA in terms of end-to-end QoS constraints (e.g. average end-to-end response time, minimum overall throughput, maximum total cost). The goal of QoS-based service composition is to select one service or service configuration for each outsourced task such that the aggregated QoS values satisfy all the application level QoS constraints.

This problem becomes especially important and challenging as the number of functionally-equivalent services offered on the web at different QoS levels increases. According to [1], there has been a more than 130% growth in the number of published web services in the period from October 2006 to October 2007. The statistics published by the web services search engine Seekda![1] also indicate an exponential increase in the number of web services over the last three years. Moreover, it is expected that with the proliferation of the Cloud Computing and Software as a Service (SaaS) concepts [5], more and more web services will be offered on the

WWW2010, April 26-30, 2010, Raleigh, North Carolina.

[1]http://webservices.seekda.com/

web at different levels of quality. The pay-per-use business model promoted by the cloud computing paradigm will enable service providers to offer their (software) services to their customers in different configurations with respect to QoS properties. Therefore, it is expected that service requesters will be soon faced with a huge number of variation of the same services offered at different QoS levels and prices, and the need for an automatic service selection method will increase.

Hence, performing an exhaustive search to find the best combination that satisfy a certain composition level SLA (i.e. end-to-end QoS constraints) is not practical in this scenario, as the number of possible combinations can be very huge, based on the number of subtasks comprising the composite process and the number of alternative services for each subtask. Already with few hundreds of candidate services (or service configurations) the required time for finding the best combination will exceed the constraints for real-time execution (e.g., with 100 alternative options for each subtask in our example, we have 100^5 possible combinations). This problem can be modeled as a combinatorial problem, which is known to be NP-hard in the strong sense, i.e. it is expected that any exact solution to this problem has an exponential cost [15]. Therefore, reducing the search space by focusing only on "interesting" service offers is crucial for reducing the computation cost.

Contributions. In this paper, we address this issue by considering dominance relationships between web services based on their QoS attributes. We observe that only those services that belong to the skyline [4], i.e. are not dominated by any other functionally-equivalent service, are valid candidates for the composition. However, although this provides an initial pruning of the number of candidate services, the size of the skyline may still be large, depending on the distribution of the QoS values. In fact, it is realistic to assume that specific QoS parameters are typically anti-correlated, e.g. execution time and price, which results in a large number of skyline services. To overcome this problem, we describe how to consider only a subset of the skyline services for the composition. In addition, from the service provider perspective, this provides also a clear distinction whether its service is a promising candidate or not for taking part in composite applications. In the latter case, we provide a strategy that proposes which QoS parameters of the service should be improved and how, so that it becomes more competitive, i.e. it is no longer dominated by other services. In particular, our main contributions can be summarized as follows.

1. We address the problem of QoS-driven service composition, defining QoS-based dominance relationships between services to select the ones to be considered for composition.
2. Since the number of candidate services for a composition may still be too large, we present a method for further reducing the search space by examining only subsets of the candidate services.
3. We present a method for determining which QoS levels of a service should be improved so that it is not dominated by other services.
4. We evaluate our approach experimentally on a publicly available collection of services with QoS information, as well as on synthetically generated scenarios.

The rest of the paper is organized as follows. Section 2 discusses related work, while section 3 introduces formally the problem. Our skyline based approach is presented in Section 4. Section 5presents our method for measuring and improving service competitiveness. The evaluation in section 6 demonstrates the benefits of our approach. Finally, section 7 concludes the paper.

2. RELATED WORK

During the last years, the problem of QoS-based web service selection and composition has received a lot of attention by many researchers. In [9] the authors propose an extensible QoS computation model that supports an open and fair management of QoS data by incorporating user feedback. However, the problem of QoS-based composition is not addressed by this work. The work of Zeng at al. [22, 23] focuses on dynamic and quality-driven selection of services. The authors use global planning to find the best service components for the composition. They use (mixed) linear programming techniques [13] to find the optimal selection of component services. Similar to this approach, Ardagna et al. [3] extend the linear programming model to include local constraints. Linear programming methods are very effective when the size of the problem is small, but suffer from poor scalability due to the exponential time complexity of the applied search algorithms [11]. In [21] the authors propose heuristic algorithms that can be used to find a near-to-optimal solution more efficiently than exact solutions. The authors propose two models for the QoS-based service composition problem: (a) a combinatorial model and (b) a graph model. A heuristic algorithm is introduced for each model. The time complexity of the heuristic algorithm for the combinatorial model (WS_HEU) is polynomial, whereas the complexity of the heuristic algorithm for the graph model (MCSP-K) is exponential. In [7], a method for Semantic Web service composition is presented, based on Genetic Algorithms and using both semantic links between I/O parameters and QoS attributes. Despite the significant improvement of these algorithms compared to exact solutions, both algorithms do not scale with respect to the number of candidate web services, and hence are not suitable for real-time service composition. The proposed skyline based algorithm in this paper is complementary to these solutions as it can be used as a pre-processing step to prune non-interesting candidate services and hence reduce the computation time of the applied selection algorithm.

In our previous work [2], we proposed a hybrid approach that combines global optimization with local selection in order to find a close-to-optimal selection efficiently. The main idea is to decompose end-to-end QoS constraints to local constraints on the component service level, which can then be used to perform efficient local selection for each component independently. The decomposition of end-to-end constraints is achieved by mapping each of them to a set of precomputed local QoS levels. In [2] we presented a greedy method for extracting QoS levels from the QoS information of service candidates. However, the proposed method deals with each QoS dimension independently and does not take potential dependencies and correlations among these dimensions into account. In some scenarios with very constrained QoS requirements, this leads to very restrictive decompositions of the global constraints to local constraints that cannot be satisfied by any of the service candidates, although a solution may actually exist. In this paper we propose a new method for extracting QoS levels, which always leads to a feasible decomposition of end-to-end constraints.

In [16], we considered dominance relationships between web services in order to rank available service descriptions with respect to a given service request. However, that work considered only selection of single services, without dealing with the problem of service composition. Moreover, in that work, dominance between services is based on their degrees of match to a given request and therefore required dynamic calculations on a per query basis. In this work our new method considers QoS-based dominance, which can be computed offline.

Finally, a first attempt to consider service competitiveness from the service provider's point of view is presented in [17]. However,

the solution presented there is simpler, considering the case where only one parameter of the service is subject to change. Instead, we propose here a more generic and flexible solution, which allows the service to improve simultaneously in more than one attributes.

3. QOS-BASED COMPOSITION MODEL

Assume a set $\mathbb{S}$ of *service classes*, which classify the universe of available web services according to their functionality. Each service class $S_j = \{s_{j_1}, ..., s_{j_n}\}, S_j \in \mathbb{S}$, consists of all web services that deliver the same functionality (e.g. used car search) but potentially differ in terms of non-functional properties. Some service providers might provide the same service in different quality levels, e.g. at different response times and different prices. For the sake of simplicity, we model each variation of the service as a different service. According to the SOA principles, descriptions of functional and non-functional properties of web services are stored and managed by service registries (e.g. UDDI registries), which are maintained by *service brokers*. In this paper, we assume that service brokers maintain and update information about existing service classes and candidate services of each class in their registries, making them accessible to service requesters.

3.1 QoS Parameters

We consider quantitative non-functional properties of web services, which can be used to describe the quality criteria of a web service. These can include generic QoS attributes like response time, availability, price, reputation etc, as well as domain-specific QoS attributes, for example bandwidth for multimedia web services, as long as these attributes can be quantified and represented by real numbers. QoS attributes may be positive or negative. The values of positive attributes need to be maximized (e.g. throughput and availability), whereas the values of negative attributes need to be minimized (e.g. price and response time). For simplicity, in this paper we consider only negative attributes (positive attributes can be easily transformed into negative by multiplying their values by -1). We use the vector $Q_s = \{q_1(s), \ldots, q_r(s)\}$ to represent the QoS values of service s, which are published by the service provider. The function $q_i(s)$ determines the published value of the i-th attribute of the service s.

3.2 QoS Computation of Composite Services

The QoS values of a composite service are determined by the QoS values of its component services and by the composition structure used (e.g. sequential, parallel, conditional and/or loops). Here, we focus on the sequential composition model. Other models may be reduced or transformed to the sequential model, using for example techniques for handling multiple execution paths and unfolding loops [6]. The QoS vector for a composite service $CS = \{s_1, \ldots, s_n\}$ is defined as Q_{CS}=$\{q'_1(CS), \ldots, q'_r(CS)\}$, where $q'_i(CS)$ is the estimated end-to-end value of the i-th QoS attribute and can be computed by aggregating the corresponding values of the component services. In our model, we consider three types of QoS aggregation functions: (1) summation, (2) multiplication and (3) minimum relation. Examples are given in Table 1.

3.3 QoS Constraints

We assume that the user has one or more requirements regarding the aggregated QoS values of the requested composite service. These requirements are expressed in terms of a vector $C = \{c_1, \ldots, c_m\}, 1 \leq m \leq r$, of upper (or lower) bounds for the different QoS criteria. We refer to them as *global QoS constraints*.

Definition 1. (Feasible Selection) For a given abstract process $P = \{S_1, \ldots, S_n\}$ and a given vector of global QoS constraints $C' = \{c'_1, \ldots, c'_m\}, 1 \leq m \leq r$, we consider a selection of concrete services CS to be a *feasible selection*, iff it contains exactly one service for each service class appearing in P and its aggregated QoS values satisfy the global QoS constraints, i.e. $q'_k(CS) \leq c'_k, \forall k \in [1, m]$.

Aggregation type	Examples	Function
Summation	Response time	$q'(CS) = \sum_{j=1}^{n} q(s_j)$
	Price	
	Reputation	$q'(CS) = 1/n \sum_{j=1}^{n} q(s_j)$
Multiplication	Availability	$q'(CS) = \prod_{j=1}^{n} q(s_j)$
	Reliability	
Minimum	Throughput	$q'(CS) = \min_{j=1}^{n} q(s_j)$

Table 1: Examples of QoS aggregation functions

3.4 Utility Function

Since each web service is typically characterized by several QoS attributes, a utility function is used to evaluate the overall, multi-dimensional quality of a given service. In particular, it maps the quality vector Q_s of the service into a single real value, to enable sorting and ranking of the alternative services. In this paper, we use a Multiple Attribute Decision Making approach for the utility function, and in particular the *Simple Additive Weighting (SAW)* technique from [20]. The utility computation involves scaling the QoS attributes' values to allow a uniform measurement of the multi-dimensional service qualities independent of their units and ranges. The scaling process is then followed by a weighting process for representing user priorities and preferences. In the scaling process, each QoS attribute value is transformed into a value between 0 and 1, by comparing it with the minimum and maximum possible value according to the available QoS information about alternative services. For a composite service CS, the aggregated QoS values are compared with minimum and maximum possible aggregated values, which can be easily estimated by aggregating, respectively, the minimum or maximum possible value of each service class in CS. For example, the maximum execution price of a given composite service can be computed by summing up the execution price of the most expensive service in each service class in CS. Formally, the minimum and maximum aggregated values of the k-th QoS attribute for a given composite service $CS = \{s_1, \ldots, s_n\}$ of an abstract process $P = \{S_1, \ldots, S_n\}$ are computed as follows:

$$Qmin'(k) = F_{j=1}^{n}(Qmin(j,k)) \qquad (1)$$
$$Qmax'(k) = F_{j=1}^{n}(Qmax(j,k))$$

with

$$Qmin(j,k) = \min_{\forall s \in S_j} q_k(s) \qquad (2)$$
$$Qmax(j,k) = \max_{\forall s \in S_j} q_k(s)$$

where $Qmin(j,k)$ is the minimum value (e.g. minimum price) and $Qmax(j,k)$ is the maximum value (e.g. maximum price) that can be expected for the k-th QoS attribute of the service class S_j, according to the available information about the service candidates in this class. The function F denotes an aggregation function that depends on QoS criteria e.g. summation, multiplication (s.a. Table 1). Now the utility of a component web service $s \in S_j$ is computed as

$$U(s) = \sum_{k=1}^{r} \frac{Qmax(j,k) - q_k(s)}{Qmax(j,k) - Qmin(j,k)} \cdot w_k \quad (3)$$

and the overall utility of a composite service is computed as

$$U'(CS) = \sum_{k=1}^{r} \frac{Qmax'(k) - q'_k(CS)}{Qmax'(k) - Qmin'(k)} \cdot w_k \quad (4)$$

with $w_k \in R_0^+$ and $\sum_{k=1}^{r} w_k = 1$ being the weight of q'_k to represent user's priorities.

3.5 Problem Statement

QoS-based service composition is a constraint optimization problem which aims at finding the composition that maximizes the overall utility value, while satisfying all the global QoS constraints. Formally:

Definition 2. (Optimal Selection) For a given abstract process P and a vector of global QoS constraints $C' = \{c'_1, \ldots, c'_m\}$, $1 \leq m \leq r$, we consider as *optimal selection* the feasible selection (see Definition 1) that maximizes the overall utility value U'.

A straightforward method for finding the optimal composition is enumerating and comparing all possible combinations of candidate services. For a composition request with n service classes and l candidate services per class, there are l^n possible combinations to be examined. Hence, performing an exhaustive search can be very expensive in terms of computation time and, therefore, inappropriate for run-time service selection in applications with many services and dynamic needs. In the following section, we present an approach to address this problem by considering dominance relationships between available services and selecting skyline services to be considered as candidates for the composite process.

4. SKYLINE SERVICES FOR QOS-BASED COMPOSITION

As presented in the previous section, the goal is to select a set of services, one from each service class, that maximize the overall utility, while satisfying all the specified constraints. Notice that, selecting from each class the service with the highest utility value does not provide a correct solution, since it does not guarantee that all the end-to-end constraints will be satisfied. Hence, different combinations of services from each class need to be considered. However, not all services are potential candidates for the solution. The main idea in our approach is to perform a skyline query on the services in each class to distinguish between those services that are potential candidates for the composition, and those that can not possibly be part of the final solution. The latter can effectively be pruned to reduce the search space. First, we briefly introduce skyline queries, and then we describe how we apply them in our approach. We also deal with the problem that arises when the number of services in the skyline is still too large.

Given a set of points in a d-dimensional space, a skyline query [4] selects those points that are not dominated by any other point. A point P_i is said to dominate another point P_j, if P_i is better than or equal to P_j in *all* dimensions and strictly better in *at least one* dimension. Intuitively, a skyline query selects the "best" or most "interesting" points with respect to *all* dimensions. In this work, we define and exploit dominance relationships between services based on their QoS attributes. This is used to identify services in a service class that are dominated by other services in the same class.

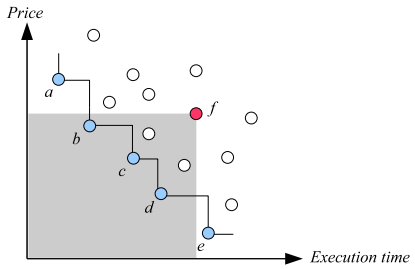

Figure 2: Example of Skyline Services

These services can then be pruned, hence reducing the number of combinations to be considered during service composition.

Definition 3. (Dominance) Consider a service class S, and two services $x, y \in S$, characterized by a set of Q of QoS attributes. x *dominates* y, denoted as $x \prec y$, 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 4. (Skyline Services) The skyline of a service class S, denoted by SL_S, comprises the set of those services in S that are not dominated by any other service, i.e., $SL_S = \{x \in S | \neg \exists y \in S : y \prec x\}$. We refer to these services as the *skyline services* of S.

Figure 2 shows an example of skyline services of a certain service class. Each service is described by two QoS parameters, namely execution time and price. Hence, the services are represented as points in the 2-dimensional space, with the coordinates of each point corresponding to the values of the service in these two parameters. We can observe that the service a belongs to the skyline, because it is not dominated by any other service, i.e. there is no other service that offers both shorter execution time *and* lower price than a. The same holds for the services b, c, d and e, which are also on the skyline. On the other hand, service f is not contained in the skyline, because it is dominated by the services b, c and d.

Notice that the skyline services provide different trade-offs between the QoS parameters, and hence are incomparable to each other, as long as there is no pre-specified preference scheme regarding the relative importance of these parameters. For instance, for a specific user, service a may be the most suitable choice, due to its very low execution time and despite its high price, while for a different user, where execution time is not the primary concern, service e may be the most preferred one due to its low price.

4.1 Determining the Skyline Services

Determining the skyline services of a certain service class requires pair-wise comparisons of the QoS vectors of the candidate services. This process can be expensive in terms of computation time if the number of candidate services is large. Several efficient algorithms have been proposed for skyline computation [14]. Given that, for the problem considered here, the process of determining the skyline services is independent of any individual service request or usage context, it does not need to be conducted online at request time. Therefore, we make use of any of the existing methods for determining the skyline services offline in order to speed up the service selection process later at request time. For this purpose, each service broker maintains the list of skyline services of each service class it hosts in its registry. This list is updated each time a service joins, leaves or updates its QoS information in the registry. When a service request is received by a service broker, the skyline services of the matched service class are returned to the requester.

If matching services are distributed over a set of service brokers, the service requester receives a skyline set from each broker. Then, the retrieved *local* skylines need to be merged to build one *global* skyline. This can be done by merging the local skylines in a pairwise fashion, i.e. comparing the services in the two local skylines, and eliminating those that are dominated by another service.

4.2 Composing the Skyline Services

Once the problem of QoS-based service composition has been formulated as a constraint optimization problem, MIP techniques [13] can be employed [23, 3]. Then, any MIP solver can be applied to solve this problem. However, as the number of variables in this model depends on the number of service candidates, it may only be solved efficiently for small instances. To cope with this limitation, we first prune all non-skyline services from the MIP model in order to keep its size as small as possible. By focusing only on the skyline services of each service class, we speed up the selection process, while still being able to find the optimal selection, as formally shown below.

Lemma 1. Given a composite service $CS = \{s_1, \ldots, s_n\}$, which is the optimal solution for a given request, i.e. the one that satisfies all the specified constraints and maximizes the overall utility. Then, each constituent service belongs to the skyline of the corresponding class, i.e. $\forall s_i \in CS : s_i \in SL_{S_i}$, where S_i denotes the class of s_i.

PROOF. Let s_i be a service that is part of CS and does not belong to the skyline of its class S_i. Then, according to the definitions for service skyline and service dominance (see Section 4), there exists another service s_i' that belongs to the skyline of S_i and dominates s_i, i.e. s_i' is better (or equal) to s_i in *all* considered QoS parameters. Let CS' be the composite service that is derived by CS by substituting s_i with s_i'. CS' also satisfies the request, in terms of the delivered functionality, since the two services s_i and s_i' belong to the same class S_i. Moreover, given that the QoS aggregation functions (see Table 1) are monotone, i.e. higher (lower) values produce a higher (lower) overall result, CS' also satisfies the constraints of the request. In addition, given that the utility function is also monotone, CS' will have a higher overall utility than CS. Hence, CS' is a better solution than CS for this request. □

According to Lemma 1, we can improve the efficiency of the QoS-based service selection algorithms by focusing only on the skyline services of each class. However, the size of the skyline can significantly vary for each dataset, as it strongly depends on the distribution of the QoS data and correlations between the different QoS parameters. Figure 3 shows an example of 3 types of datasets in the 2-dimensional space: (a) in the independent dataset, the values on the two QoS dimensions are independent to each other; (b) in the correlated dataset, a service that is good in one dimension is also good in the other dimension; (c) in the anti-correlated dataset there is a clear trade-off between the two dimensions. The number of skyline services is relatively small in correlated datasets, large in anti-correlated and medium in independent ones.

In the cases that the skyline of a dataset is too large to be practically useful, approaches have been proposed for limiting the selection to a number of representative items [8, 19]. In [18] we have investigated such an approach for web service discovery in order to cluster the matched services returned to the user. Here, our goal is to select a set of representative skyline services, providing different trade-offs for the various QoS parameters, and use this reduced set as input for the MIP model.

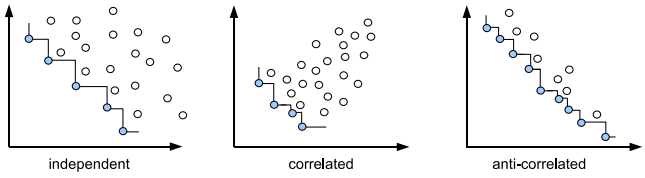

Figure 3: Skyline of Different Dataset Types

4.3 Representative Skyline Services

In the following, we present a method for selecting representative skyline services in order to address the situation where the number of skyline services K of a certain service class S is too large and thus cannot be handled efficiently. The main challenge that arises is how to identify a set of representative skyline services that best represent all trade-offs of the various QoS parameters, so that it is possible to find a solution that satisfies the constraints and has also a high utility score. This involves essentially a trade-off regarding the number of representatives to be selected: the number of representative services should be large enough to allow finding a solution to the composition request, but also small enough to allow for efficient computation.

To address this challenge, we propose a hierarchical clustering-based method. The main idea is to cluster the skyline services into k clusters with $k = 2, 4, 8, 16, ..., K$ and select one representative service from each cluster. In our case, we select as representative the service with the best utility value. In particular, we build a tree structure of representatives, as shown in the example of Figure 4. Each leaf node of this tree corresponds to one of the skyline services in SL, whereas the root and intermediate nodes correspond to the selected representatives of the created clusters.

At run-time, when a service composition request is processed, we start the search from the root node of the tree, i.e. we first consider only the top representative service of each class (e.g. service s_3 for class S in the example). These selected representatives are inserted into the mixed integer program and the optimization problem is solved. In the case that no solution is found using the given representatives, we proceed to the next level, taking two representatives for each class (s_3 and s_6 for class S in the example). This process is repeated until a solution is found or until the lowest level of the tree, which consists of all skyline services, is reached. In the latter case, it is guaranteed that a solution will be found (if one exists), and this solution is the optimal solution according to Lemma 1. However, if a solution is found earlier, i.e. before reaching the skyline level, we proceed by examining those services that are descendants of the selected representatives for further optimization. This expanding of the search space is continued until no further optimization in terms of utility value is achieved, or the skyline level is reached.

We use the well-known *k-means* clustering algorithm [10] for building the representatives tree, as described in Algorithm 1. The algorithm takes as input the skyline set SL of class S and returns a binary tree structure of representative services. The algorithm starts by determining the root s, which is the service with maximum utility value in SL. The algorithm then clusters SL into two sub-clusters $CLS[0]$ and $CLS[1]$ and adds the representatives of these two sub-clusters to the children list of s. The process is then repeated for each sub-cluster until no further clustering is possible (i.e. until the size of new created clusters is lower than 2).

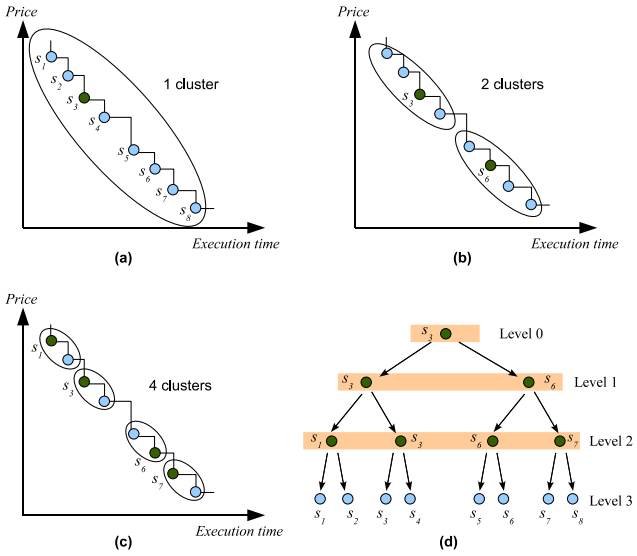

Figure 4: Determining Representatives via Hierarchical Clustering

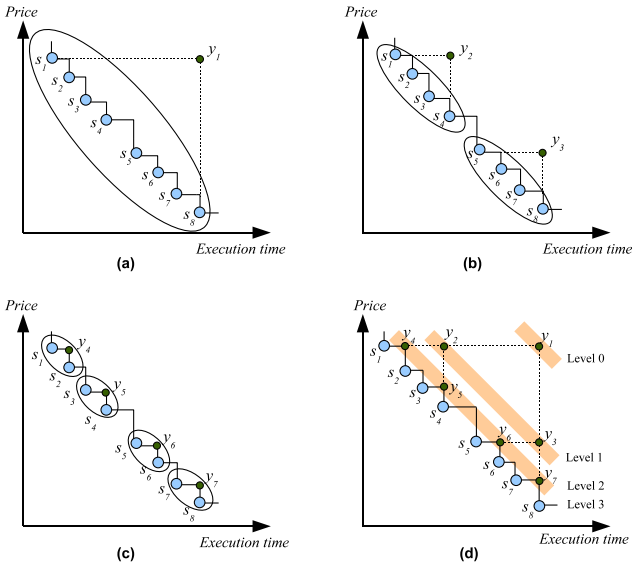

Figure 5: Determining Local Quality Levels

Algorithm 1 BuildRepresentativesTree(SL)

Input: : a set of skyline services SL
Output: : a tree of representatives with service s as a root

1: $s \leftarrow maxUtilityService(SL)$
2: $CLS \leftarrow KMeansCluster(SL, 2)$
3: **for** $i = 1$ **to** 2 **do**
4: **if** $(CLS[i].size > 2)$ **then**
5: $C \leftarrow BuildRepresentativesTree(CLS[i])$
6: **else**
7: $C \leftarrow CLS[i]$
8: **end if**
9: $s.addChild(C)$
10: **end for**
11: **return** s

Algorithm 2 SelectQoSLevels(SL)

Input: : a set of skyline services SL
Output: : a tree of QoS levels with y as a root

1: $y \leftarrow newQoSLevel$
2: **for all** $q_i \in Q$ **do**
3: $q_i(y) \leftarrow \max q_i(s), \forall s \in SL$
4: **end for**
5: $y.utility \leftarrow maxUtilityValue(SL)$
6: $CLS \leftarrow KMeansCluster(SL, 2)$
7: **for** $i = 1$ **to** 2 **do**
8: **if** $(CLS[i].size > 2)$ **then**
9: $C \leftarrow SelectQoSLevels(CLS[i])$
10: **else**
11: $C \leftarrow CLS[i]$
12: **end if**
13: $y.addChild(C)$
14: **end for**
15: **return** y

4.4 Local QoS Levels

So far, we have described how the efficiency of the standard MIP-based global optimization approach for QoS-based web service composition can be improved by focusing on the representative skyline services of each service class. In [2], we proposed a hybrid approach for solving this problem, using MIP to decompose the end-to-end QoS constraints into local constraints, which are then used to efficiently select the best service from each class. The variables in the MIP model of the hybrid approach represent the local QoS levels of each service class rather than the actual service candidates, making it more scalable to the number of service candidates than the global optimization approach. However, the proposed solution in [2] relies on a greedy method for extracting the local QoS levels from the QoS information of service candidates, which deals with QoS parameters independently and does not take into account potential correlations and dependencies among them. Hence, in scenarios with relatively strict constraints, this often leads to a very restrictive decomposition of the constraints that cannot be satisfied by any of the service candidates even though a solution to the problem does exist.

To overcome the limitation of that method, we present in the following a new method for extracting QoS levels, which always leads to a feasible decomposition of end-to-end constraints based on skyline services (see Algorithm 2). The main idea is similar to the representatives selection method described earlier. First, we determine the skyline services of each service class, and we recursively cluster them using the k-means clustering algorithm. However, instead of selecting one representative service from each sub-cluster, we create a virtual point in the QoS multidimensional space, whose coordinates are calculated as the maximum (i.e. worst) QoS values in the sub-cluster, as illustrated in the example of Figure 5. The virtual point y_1 in Figure 5-a has the maximum execution time and maximum price of all skyline services, i.e the execution time of service s_8 and the price of service s_1.

Hence, we use the created points (y_1 to y_7 in the example) to represent the various QoS levels of the service class. We also assign each of the QoS levels a utility value, which is the best utility value that can be obtained by any of the services of the corresponding sub-cluster. We then use MIP to map each of the end-to-end constraints into one of the local QoS levels of each class in the composition problem. A binary decision variable x_{ij} is used for each local QoS level y_{ij} such that $x_{ij} = 1$ if y_{ij} is selected as a

local constraint for the service class S_j, and $x_{ij} = 0$ otherwise. Thus, we reformulate the MIP model presented in [2] as follows:

$$\text{maximize} \quad \sum_{j=1}^{n}\sum_{i=1}^{l} U(y_{ij}) \cdot x_{ij} \tag{5}$$

subject to the global QoS constraints

$$\sum_{j=1}^{n}\sum_{i=1}^{l} q_k(y_{ij}) \cdot x_{ij} \leq c'_k, 1 \leq k \leq m \tag{6}$$

while satisfying the allocation constraints on the decision variables as

$$\sum_{i=1}^{l} x_{ij} = 1, 1 \leq j \leq n. \tag{7}$$

where the number of variables l equals the number of QoS level in each service class. We solve this MIP model for $l = 1, 2, 4....K$, where K is the total number of skyline services. In the given example, this corresponds to the levels from 0 to 3 of the QoS levels tree in Figure 5-d. The process stops when a solution is found, i.e. a mapping of all end-to-end constraints to local QoS levels is found. In the worst case, the process will continue until the lowest level is reached. In this case, each skyline service represents a local QoS level, and the problem becomes similar to the original global optimization problem we discussed earlier. According to Lemma 1, if a solution to the original problem exists, a decomposition of the end-to-end constraints will be found.

5. IMPROVING SERVICE COMPETITIVENESS

As described in the previous section, when a service composition request is processed, only the skyline services of each participating service class are examined as possible candidates. Hence, a non-skyline service is filter out early, and it cannot be on the result set of any potential request, regardless of the given QoS requirements or preferences. Therefore, it is important for service providers to know whether their services are on the skyline, given their current QoS levels. Even more importantly, if this is not the case, the providers should be guided in determining which QoS levels of their services should be improved and how, in order to become skyline services. Such information can be very valuable for service providers as it enables them to analyze the position of their services on the market compared to other competing services.

To address this issue, we present an algorithm for assisting providers of non-skyline services in improving the competitiveness of their services. Clearly, there are various modifications that can lead to a non-skyline service becoming part of the skyline. Our goal is to find the minimum modification of the service's QoS values that is sufficient for constituting this service a skyline service. More specifically, we propose an algorithm that identifies the minimum improvement in each QoS dimension that is required in order to bring a non-skyline service into a position where it is not dominated by any other service, thus becoming part of the skyline.

Consider the example shown in Figure 6, where a service f is dominated by the skyline services b, c and d. According to definition (3), this means that each of these services are better or equal to f in all QoS dimensions and strictly better than f in at least one QoS dimension. Therefore, service f cannot be part of the result set of any composition request comprising this service class. In order to improve the competitiveness of service f, the provider must ensure that it is not dominated by any other service. To achieve this,

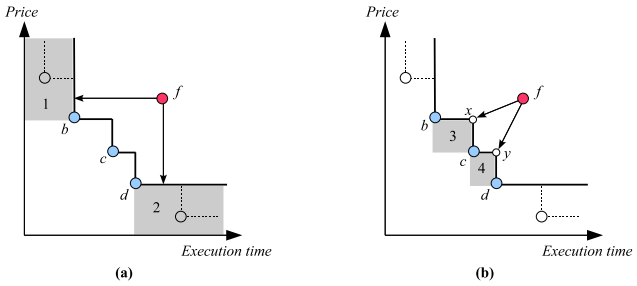

Figure 6: Measuring the Distance to the Skyline

it is sufficient to make service f better than each of its dominating services in (at least) one QoS dimension. By analyzing the skyline structure in Figure 6, we can identify four partitions of the 2-dimensional space, in which service f can fulfill this requirement. The first two partitions are shown in Figure 6-a, and can be reached by improving only one of the QoS-dimensions, while the other two are shown in Figure 6-b, and can be reached by improving both QoS dimensions at the same time. We call each of these partitions a *no-dominance* partition for this service. A service in any of these partitions is incomparable with all the skyline services, as it is not dominated by any of them nor is dominating any of them.

Improving the QoS of provided services to a certain level typically imposes some overhead. For example, reducing the execution time of the service might require buying faster servers or more CPU computation power, if the service is running on the cloud. Thus, service providers would be interested in determining the best (set of) QoS dimension(s) to optimize, while minimizing the required cost. We assume that the cost of improving any QoS dimension increases monotonically in the sense that more improvement always implies more cost. We use the *weighted euclidean distance* for estimating the cost of moving a service s in the QoS multi-dimension space from its current position p_1 to a new position p_2:

$$d_{p_1,p_2}(s) = \sqrt{\sum_{k=1}^{r} w_k(p_1(k) - p_2(k))^2} \tag{8}$$

The weight w_k is specified by the service provider to express his preferences over the QoS dimensions. A higher weight implies higher cost for improving the corresponding QoS dimension.

In order to minimize the cost of improving the service position in the QoS multi-dimensional space, we first need to identify the no-dominance partitions. Then, we measure the distance from the service to be improved to each of these partitions using equation 8, and we select the one with the minimum distance.

Algorithm 3 locates the no-dominance partitions that can be reached by improving only one QoS dimension. The algorithm takes as input a non-skyline service s and the list of skyline services SL of the corresponding class, and it returns a list $I = \{p_1, \ldots, p_{|Q|}\}$, where each entry p_i denotes the improvement required in the i-th QoS dimension in order the service to become part of the skyline (keeping all the other dimensions fixed).

Algorithm 4 is used to locate the coordinates of the maximum corner (i.e. top-right) of each no-dominance partition (i.e. the points *x* and *y* in Figure 6-b). Modifying the QoS values of service f to values that are slightly better than the values of one of these points, ensures that f is not dominated by the skyline services. The algorithm takes as input a non-skyline service s and the list of skyline services SL of the corresponding class, and it

Algorithm 3 OneDimImprovements(s, SL)

Input: : a service s, the skyline services of that class SL
Output: : a list I containing the required improvement for each single dimension

1: $DS \leftarrow \{r \in SL : r \succ s\}$
2: **for all** $q_i \in Q$ **do**
3: $\quad I[i] \leftarrow \max_{r \in DS} |r^{q_i} - s^{q_i}|$
4: **end for**
5: **return** I

returns a list M of maximum corners of no-dominance partitions. First, it computes the list DS, which comprises the services dominating s. Then, DS is sorted for each QoS dimension separately. The coordinates of the maximum corners are determined by taking the maximum QoS values of each two subsequent services in each sorted list. The coordinates of the maximum corners x and y in Figure 6-b, for example, are determined by sorting the dominating services b, c and d by execution time and then taking the maximum price and execution time of the services b and c. This process is then repeated for each other dimension and only new discovered points are added to the list M.

Algorithm 4 MultiDimImprovements(s, SL)

Input: : a service s, the skyline services of that class SL
Output: : a set M of maximum corners of no-dominance partitions

1: $DS \leftarrow \{r \in SL : r \succ s\}$
2: $M \leftarrow \{\}$
3: **for all** $q_i \in Q$ **do**
4: $\quad DS_i \leftarrow DS.sortBy(q_i)$
5: $\quad$ **for** $j = 1$ **to** $DS.size - 1$ **do**
6: $\quad\quad s_j \leftarrow DS_i[j]$
7: $\quad\quad s_{j+1} \leftarrow DS_i[j+1]$
8: $\quad\quad m \leftarrow newQoSVector$
9: $\quad\quad$ **for all** $q_i \in Q$ **do**
10: $\quad\quad\quad q_i(m) \leftarrow \max(q_i(s_j), q_i(s_{j+1}))$
11: $\quad\quad$ **end for**
12: $\quad\quad$ **if** $m \notin M$ **then**
13: $\quad\quad\quad M \leftarrow m$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: **end for**
17: **return** M

6. EXPERIMENTAL EVALUATION

In this section, we present an experimental evaluation of our approach, focusing on its efficiency, in terms of the execution time required to find a solution, and the success rate, in terms of whether a solution is found (if one exists) and how close its utility score is compared to that of the optimal solution.

6.1 Experimental Setup

In our evaluation we experimented with two types of datasets. The first is the publicly available dataset QWS[2], which comprises measurements of 9 QoS attributes for 2500 real-world web services. These services were collected from public sources on the Web, including UDDI registries, search engines and service portals, and their QoS values were measured using commercial benchmark tools. More details about this dataset can be found in [1]. We also experimented with three synthetically generated datasets in order to test our approach with larger number of services and different distributions. For this purpose, we used a publicly available synthetic generator[3] to obtain three different datasets: a) a correlated dataset (cQoS), in which the values of the QoS parameters are positively correlated, b) an anti-correlated(aQoS) dataset, in which the values of the QoS parameters are negatively correlated, and c) an independent dataset, in which the QoS values are randomly set. Each dataset comprises 10K QoS vectors, and each vector represents the 9 QoS attributes of one web service.

For the purpose of our evaluation, we considered a scenario, where a composite application comprises services from 10 different service classes. Thus, we randomly partitioned each of the aforementioned datasets into 10 service classes. We then created several QoS vectors of up to 9 random values to represent the users end-to-end QoS constraints. Each QoS vector corresponds to one QoS-based composition request, for which one concrete service needs to be selected from each class, such that the overall utility value is maximized, while all end-to-end constraints are satisfied.

We implemented the algorithms described in Section 4 in Java. For solving the generated Mixed Integer Programming models we used the open source system *lpsolve* version 5.5 [12]. The experiments were conducted on an HP ProLiant DL380 G3 machine with 2 Intel Xeon 2.80GHz processors and 6 GB RAM, running Linux (CentOS release 5).

We compare the efficiency of the following QoS-based composition methods:

- *Exact*: this is the standard global optimization method with all service candidates represented in the MIP model.
- *ExactSykline*: this method is similar to the Exact method, except that only skyline services are considered here.
- *SkylineRep*: this method uses representative skyline services as described in Section 4.3.
- *Hybrid*: this is the method we proposed in our previous work [2], which maps end-to-end constraints into local QoS levels.
- *HybridSkyline*: this is the modified version of the Hybrid method, which uses a skyline-based method for determining local QoS levels as described in Section 4.4.

6.2 Performance vs Number of Services

We measured the average execution time required by each of the aforementioned methods for solving each composition problem, varying the number of service candidates from 100 to 1000 services per class. The results of this experiment are presented in Figure 7.

Comparing the performance of *Exact* and *ExactSkyline* methods, we can observe that a significant gain is achieved when non-skyline services are pruned. However, as expected, this gain in performance differs for the different datasets, based on the size of the skyline, with the lowest gain being recorded for the anti-correlated dataset. On the other hand, the *SkylineRep* method clearly outperforms all other methods, which shows that we can cope effectively with this limitation by using skyline representatives as described in Section 4.3. In general, the performance of the *HybridSkyline* method is comparable with the performance of the *Hybrid* method as long as the size of the skyline is not very large (see the performance of both methods with the QWS and correlated datasets).

[2]http://www.uoguelph.ca/ qmahmoud/qws/index.html/

[3]http://randdataset.projects.postgresql.org/

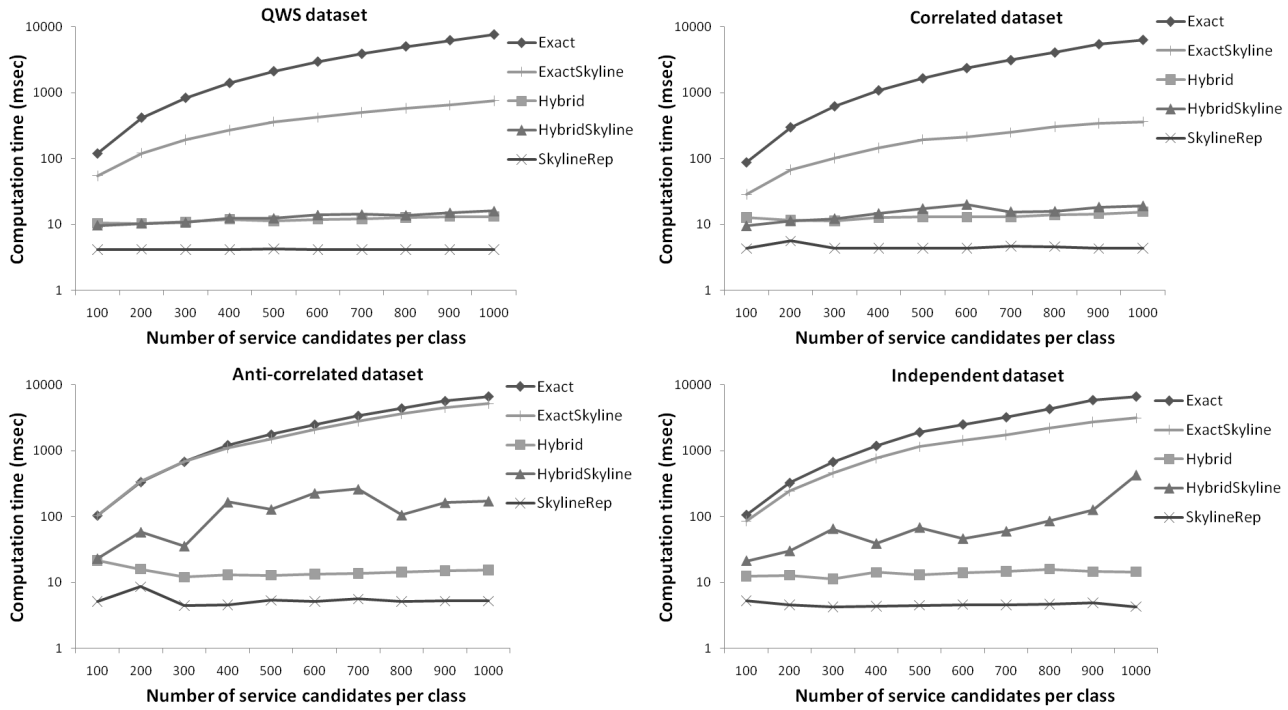

Figure 7: Performance vs. Number of Service Candidates

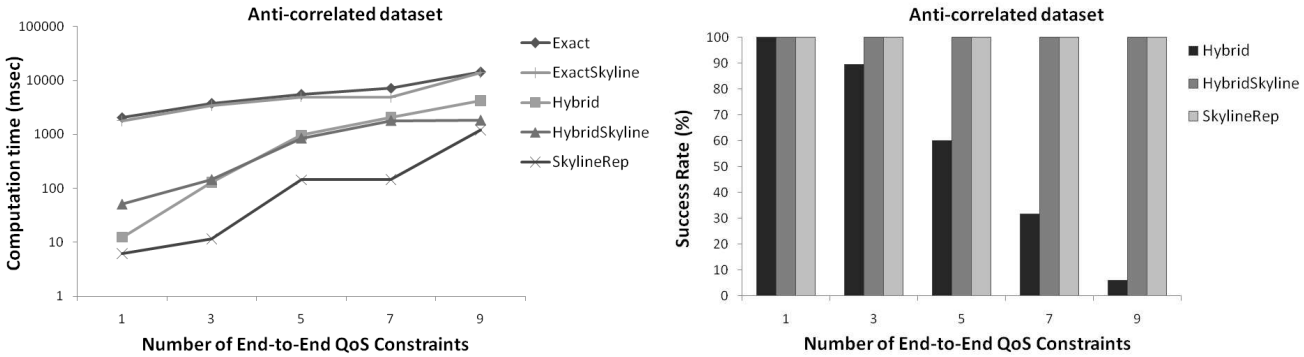

Figure 8: Performance and Success Rate vs. QoS Constraints

Although less efficient than the original *Hybrid* method with the independent and anti-correlated datasets, the *HybridSkyline* method still outperforms the *Exact* method with more than an order of magnitude gain in performance. Moreover, the *HybridSkyline* method outperforms the *Hybrid* method in terms of success rate as we will see in the next subsection.

We also computed the optimality of the returned selection by comparing the overall utility value u of the selected services with the overall utility value (u_{exact}) of the optimal selection obtained by the Exact approach, i,e.:

$$optimality = u/u_{exact}$$

The measured optimality of the *SkylineRep*, *Hybrid* and *HybridSkyline* methods was in all cases above 90%, which indicates the ability of these methods to achieve close-to-optimal results.

6.3 Performance vs Number of QoS Constraints

Clearly, the number of feasible selections for a given composition request decreases as the number of end-to-end QoS constraints increases. This can affect the performance of all methods as more computation time is required to find a solution. More specifically, with very constrained problems the probability that the iterative algorithm of *SkylineRep* and *HybridSkyline* will need to go through more iterations until a solution is found increases. In this experiment we measured the performance of the different methods with respect to the number of end-to-end QoS constraints. For this purpose, we fixed the number of service candidates per class to 500 services, and we varied the number of QoS constraints from 1 to 9 (notice that the total number of QoS parameters in the QWS dataset is 9). Due to space limitations, in Figure 8 we only show the results of this experiment with the anti-correlated dataset, as this dataset represents the most challenging scenario, due to the large size of the skyline. Again, we observe that *SkylineRep* clearly outperforms all other approaches. The *Hybrid* and *HybridSkyline* methods have similar performance, also outperforming the *Exact* solution. In addition, we measured the success rate, i.e., the percentage of scenarios where a solution is found, if one exists (see the right-side graph

in Figure 8). As shown, *SkylineRep* and *HybridSklyine* always find a solution. This is because *SkylineRep* and *HybridSklyine* iteratively expand the search space by examining more representative services or local QoS levels,respectively, until a solution is found or until the whole set of skyline services has been examined. In the latter case, a solution is guaranteed to be found (if one exists) according to Lemma 1. On the other hand,the success rate of the *Hybrid* method degrades significantly as the difficulty of the composition problem increases. The reason for this behavior is that the *Hybrid* method decomposes each of the end-to-end constraints independently, which in such difficult composition problems may results in a set of local constraints that cannot be satisfied by any candidate.

7. CONCLUSIONS

In this paper, we have addressed the problem of QoS-based web service composition. We identify and exploit the skyline services in terms of their QoS values, and we have proposed an algorithm that improves the efficiency of the state-of-the-art solution by pruning non-skyline services. Moreover, to deal with cases where the size of the skyline is still large compared to the initial dataset, we have proposed a method to select and use representative skyline services for the composition. We have also presented an effective method for determining local quality levels, and hence, which improves the success rate of the hybrid solution for QoS-based service composition from our previous work [2]. Finally, we have presented a method for assisting service providers in improving the competitiveness of their services to attract potential clients. The results of the experimental evaluation indicate a significant performance gain in comparison to existing approaches, which rely on global optimization.

Our experiments have also shown that the performance of our skyline-based methods is affected by the difficulty of the composition problem, in terms of the number and strength of the specified end-to-end QoS constraints. To overcome this problem, in the future work, we plan to develop a method for estimating the difficulty of each composition problem. Then, based on the difficulty level of the problem, we can decide from which level of the skyline representatives tree (or the QoS levels tree) to start the search, i.e, lower in the case of stricter constraints, in order to avoid unnecessary iterations and have a smaller increase in the execution time.

8. REFERENCES

[1] E. Al-Masri and Q. H. Mahmoud. Investigating web services on the world wide web. In *International World Wide Web Conference*, 2008.

[2] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *International World Wide Web Conference*, pages 881–890, 2009.

[3] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. on Software Engineering*, 33(6):369–384, 2007.

[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *International Conference on Data Engineering*, pages 421–430, 2001.

[5] K. S. Candan, W.-S. Li, T. Phan, and M. Zhou. Frontiers in information and software as services. In *International Conference on Data Engineering*, pages 1761–1768, 2009.

[6] J. Cardoso, J. Miller, A. Sheth, and J. Arnold. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 1:281–308, 2004.

[7] F. Lecue. Optimizing qos-aware semantic web service composition. In *ISWC*, 2009.

[8] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *International Conference on Data Engineering*, pages 86–95, 2007.

[9] Y. Liu, A. H. H. Ngu, and L. Zeng. Qos computation and policing in dynamic web service selection. In *International World Wide Web Conference*, pages 66–73, 2004.

[10] S. P. Lloyd. Least squares quantization in pcm. *IEEE Trans. on Information Theory*, 28:129–137, 1982.

[11] I. Maros. *Computational Techniques of the Simplex Method*. Springer, 2003.

[12] P. N. Michel Berkelaar, Kjell Eikland. Open source (mixed-integer) linear programming system. Sourceforge. `http://lpsolve.sourceforge.net/`.

[13] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY, USA, 1988.

[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. on Database Systems*, 30(1):41–82, 2005.

[15] R. Parra-Hernandez and N. J. Dimopoulos. A new heuristic for solving the multichoice multidimensional knapsack problem. *IEEE Trans. on Systems, Man, and Cybernetics, Part A*, 35(5):708–717, 2005.

[16] D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. Sellis. Top-k dominant web services under multi-criteria matching. In *Extending Database Technology*, pages 898–909, 2009.

[17] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis. Serving the sky: Discovering and selecting semantic web services through dynamic skyline queries. In *International Conference on Semantic Computing*, pages 222–229, 2008.

[18] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis. Ranking and clustering web services using multi-criteria dominance relationships. In *IEEE Trans. on Services Computing (under revision)*, 2009.

[19] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *International Conference on Data Engineering*, pages 892–903, 2009.

[20] K. . P. Yoon and C.-L. Hwang. *Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences*. Sage Publications, 1995.

[21] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. on the Web*, 1(1), 2007.

[22] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *International World Wide Web Conference*, pages 411–421, 2003.

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