A Hybrid Approach for Efficient Web Service Composition with End-to-End QoS Constraints

MOHAMMAD ALRIFAII, L3S Research Center and University of Hanover
THOMAS RISSE, L3S Research Center and University of Hanover
WOLFGANG NEJDL, L3S Research Center and University of Hanover

Dynamic selection of web services at run-time is important for building flexible and loosely-coupled service-oriented applications. An abstract description of the required services is provided at design-time and matching service offers are located at run-time. With the growing number of web services that provide the same functionality but differ in quality parameters (e.g., availability, response time), a decision needs to be made on which services should be selected such that user’s end-to-end QoS requirements are satisfied. Although very efficient, local selection strategy fails short in handling global QoS requirements. Solutions based on global optimization, on the other hand, can handle global constraints, but their poor performance renders them inappropriate for applications with dynamic and real-time requirements. In this paper we address this problem and propose a hybrid solution that combines global optimization with local selection techniques to benefit from the advantages of both worlds. The proposed solution consists of two steps: first, we use mixed integer programming (MIP) to find the optimal decomposition of global QoS constraints into local constraints. Second, we use distributed local selection to find the best web services that satisfy these local constraints. The results of experimental evaluation indicate that our approach significantly outperforms existing solutions in terms of computation time while achieving close-to-optimal results.

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

Additional Key Words and Phrases: Web Services, QoS, Optimization, Service Composition

ACM Reference Format:

1. INTRODUCTION

The service-oriented computing paradigm and its realization through standardized web service technologies provide a promising solution for the seamless integration of business applications to create new value-added services. Industrial practice witnesses a growing interest in the ad-hoc service composition in the areas of supply chain management, accounting, finances, eScience as well as in multimedia applications. With the growing number of alternative web services that provide the same functionality but differ in quality parameters, the composition problem becomes a decision problem on the selection of component services with regards to functional and non-functional requirements.

Figure 1 gives a conceptual overview of the web service composition. At design-time the provider of the composite service defines the set of required services and structure them in a way that best fulfill the overall goal. A workflow-like language such as WS-BPEL [OASIS 2007] or YAWL [van der
Aalst and ter Hofstede [2005] is usually used to model the abstract representation of the composite service. Next, in a pre-processing step, service discovery is performed by exploiting the existing infrastructure (e.g. UDDI) to locate available web services for each task in the workflow using syntactic (and probably semantic) functional matching between the tasks and service descriptions. As a result, a list of functionally-equivalent web services (referred to as candidate services) is obtained for each task.

At run-time, QoS-aware service selection is performed upon service request in order to select one component service from each list of candidate services such that the aggregated QoS values satisfy the user’s QoS requirements. Users of composite applications are typically unaware of the involved services, and they specify their QoS requirements in terms of end-to-end QoS constraints (e.g. average end-to-end response time, minimum overall throughput, maximum total cost etc). Selecting the best service from a list of alternative services for each task such that all user’s QoS requirements are satisfied, is a non-trivial task as the number of possible combinations can be very huge. This problem is an instance of combinatorial problem, the Multi-dimensional Multiple-choice Knapsack Problem (MMKP), which is known to be NP-hard in the strong sense. Any exact solution to this problem is expected to have an exponential computational complexity with respect to the number of candidate services, which can be out of the run-time requirements.

In service oriented environments with real business settings, the efficiency of the applied selection mechanism becomes crucial. The focus of this paper is on the selection of web services based on their QoS values and the performance of the applied selection techniques.

**Contribution of the Paper**

Since the QoS requirements (e.g. response times, throughput or availability) are only approximate, we argue that finding a “reasonable” set of services that avoid obvious violations of constraints at acceptable costs is more important than finding “the optimal” set of services with a very high cost. In addition, we advocate that the selection of component services should be carried out in a
distributed fashion, which fits well to the open web service environment, where central management is not feasible.

The work in this paper extends and complements our previous work published in [Alrifai and Risse 2009] by giving more details on the approach including the QoS computation of complex composite services, by presenting an improved method for the local selection of quality levels, and by providing a more in depth evaluation of the whole approach. The contribution of this paper can be stated as follows:

— **A distributed QoS computation model for web services.** Unlike existing solutions that model the QoS-aware service composition problem as a conventional global optimization problem, we exploit the special structure of the web service composition problem to reduce the cost of QoS optimization. The QoS optimization in our model is carried out by a set of distributed service brokers. The idea is to decompose QoS global constraints into a set of local constraints that will serve as a conservative upper/lower bounds, such that the satisfaction of local constraints by a local service broker guarantees the satisfaction of the global constraints.

— **An efficient QoS-aware service selection approach.** We propose an efficient and scalable mechanism for selecting web services for a given composition request from a collection of service candidates, such that the fulfillment of user’s end-to-end QoS requirements and preferences can be ensured. By combining global optimization with local selection our approach is able to efficiently solve the selection problem in a distributed manner. In this paper we present an improved method for selecting quality levels that follows an iterative approach to ensure that for almost all cases an efficient and successful selection of services can be found.

— Different than the initial work presented in [Alrifai and Risse 2009], which focuses on sequential compositions, this work will generalize the proposed computation model in order to handle more complex composition structures. This generalization gives an additional advantage to the hybrid approach compared to the global approach, which cannot be directly applied to non-sequential compositions.

— **Extended experimental evaluations** that investigate show the scalability of our hybrid approach with respect to different parameters. Beside the verification that our approach is able to reach close-to-optimal results much faster than existing ‘pure’ global optimization approaches, we will evaluate the impact of the new iterative quality level selection method on the results. We also compare the performance of the hybrid approach with the performance of the WS-HEU algorithm proposed in [Yu et al. 2007]. In addition to the real world data set we used in [Alrifai and Risse 2009], we also will evaluate extreme cases by using synthetic generated correlated, anti-correlated and independent dataset.

The rest of the papers is organized as follows. In the next section we discuss related work. Section 3 introduces the system model and gives a problem statement. Our approach for efficient and distributed QoS-aware service selection is presented in Section 4. Performance analysis and experimental evaluations for comparing our solution against existing solutions are presented in Section 5. Finally, Section 6 gives conclusions and an outlook on possible continuations of our work.

2. RELATED WORK

Quality of Service management has been widely discussed in the area of middleware systems [Au-recoechea et al. 1998; Casati and Shan 2001; Cui and Nahrstedt 2001; Gillmann et al. 2002]. Most of these works focus on QoS specification and management. Recently, the QoS-based web service selection and composition in service-oriented applications has gained the attention of many researchers [Zeng et al. 2003; Zeng et al. 2004; Liu et al. 2004; Ardagna and Pernici 2005; 2007; Yu et al. 2007; Kritikos and Plexousakis 2009; Zhai et al. 2009]. In [Zhou et al. 2004; Bilgin and Singh 2004] ontology-based representations for describing QoS properties and requests were proposed to support semantic and dynamic QoS-based discovery of web services. In [Liu et al. 2004] the authors propose an extensible QoS computation model that supports open and fair management of QoS data.
Two general approaches exist for the QoS-aware service composition: local selection and global optimization.

**Local Selection.** The local selection approach is especially useful for distributed environments where central QoS management is not desirable and groups of candidate web services are managed by distributed service brokers [Benatallah et al. 2002; Li et al. 2007]. The idea is to select one service from each group of service candidates independently on the other groups. Using a given utility function, the values of the different QoS criteria are mapped to a single utility value and the service with maximum utility value is selected. This approach is very efficient in terms of computation time as the time complexity of the local optimization approach is $O(l)$, where $l$ is the number of service candidates in each group. Even if the approach is useful in decentralized environments, local selection strategy is not suitable for QoS-based service composition, with end-to-end constraints (e.g. maximum total price), since such global constraints cannot be verified locally.

**Global Optimization.** The global optimization approach was put forward as a solution to the QoS-aware service composition problem [Zeng et al. 2003; Zeng et al. 2004; Ardagna and Pernici 2005; 2007; Kritikos and Plexousakis 2009]. This approach aims at solving the problem on the composite service level. The work of Zeng et al. [Zeng et al. 2003; Zeng et al. 2004] 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 Integer Programming techniques [Nemhauser and Wolsey 1988] (MIP) to find the optimal selection of component services. Similar to this approach Ardagna et al. [Ardagna and Pernici 2005; 2007] extend the linear programming model to include local constraints. In their model, global constraints are specified by the end user on the composition level, while local constraints can be specified by the designer of the composition on the component services’ level. Unlike this approach, our proposed solution decomposes all end user’s global constraints into local constraints. Our solution can also easily handle local constraints given by the designer of the composition. Another difference between the two approaches, is that in our approach, mixed integer programming is applied for the decomposition of the constraints not for the selection of the services. As we discuss later in Section 4 and Section 5, the number of random variables in our model is much smaller than the number of random variables in the other approach, which makes our model more efficient in terms of computation time.

In [Kritikos and Plexousakis 2009] Kritikos and Plexousakis claim that mixed-integer programming should be used as a matchmaking technique instead of Constrained Programming (CP) and provide experimental results proving it. Zhai et al. [Zhai et al. 2009] propose a solution for repairing failed service compositions by replacing failed services only and reconfiguring the composition in a way that still meets the user’s end-to-end QoS requirements. The reconfiguration of the composition and the suggestion of new services is based on MIP. Generally, MIP methods are very effective when the size of the problem is small. However, these methods suffer from poor scalability due to the exponential time complexity of the applied search algorithms [Maros 2003]. Already in larger enterprises and even more in open service infrastructures with a few thousands of services the response time for a service composition request could already be out of the real-time requirements.

**Heuristic Solutions:** As discussed earlier, the problem of QoS-aware service selection can be modeled as a Multi-dimensional Multiple-choice Knapsack Problem (MMKP). In MMKP problem, a set of groups of items, where each item has a profit value and consumes some resources exist. The goal of this problem is to select exactly one item from each group such that the total profit value is maximized under some constraints on the total resource consumptions. The groups and items in this problem correspond to the service classes and the candidate services in the web service scenario respectively. The profit value of an item corresponds to the utility value of a web service and the constraints on the resource consumption correspond to the QoS constraints.
There exist a number of heuristics in the literature for solving the Knapsack Problem in general and the MMKP variant of this problem in particular. In [Khan 1998] a heuristic named HEU for solving the MMKP was presented. HEU uses a measurement called aggregate resource consumption to decide upon which item from each group should be upgraded in each round of selection. In [Akbar et al. 2001] a modified version of HEU named M-HEU was presented, where a pre-processing step to find a feasible solution and a post-processing step to improve the total value of the solution with one upgrade (i.e., item selection that increases the total profit value) followed by one or more downgrades (i.e., item selection that decreases the total profit value) were added. In [Akbar et al. 2006] the authors propose another heuristic for solving the MMKP named C-HEU and evaluate its performance and optimality against several heuristics including the M-HEU algorithm. The results of their evaluation show that C-HEU outperforms M-HEU in terms of computation time. However, the experiments have also shown that M-HEU produces the nearest to the optimal solution among all the heuristics, while the optimality of C-HEU decreases as the number of items in each group increases. Furthermore, the results have shown that the C-HEU algorithm performs better in systems, where the objective value to be maximized (i.e., the utility value in the web service scenario) is not proportional to the resource requirements (i.e., the QoS values of web services). Since the utility value of a given web service is proportional to the QoS level of the service, the C-HEU algorithm is not applicable to the QoS-aware service selection problem.

A modified version of the M-HEU algorithm named WS-HEU, designed for the QoS-aware service selection problem was proposed in [Yu et al. 2007]. The authors propose two models for the QoS-based service composition problem: 1) a combinatorial model and 2) a graph model. A heuristic algorithm is introduced for each model: the WS-HEU algorithm for the combinatorial model and the MCSP-K for the graph model. The time complexity of WS-HEU is polynomial, whereas the complexity of MCSP-K is exponential. Despite the significant improvement of these algorithms compared to exact solutions, both algorithms do not scale with respect to an increasing number of web services and remain out of the real-time requirements. In our experimental evaluation, which we present in Section 5.2 we compare our hybrid approach against the WS-HEU algorithm. The results indicate the the hybrid approach outperforms the WS-HEU.

Moreover, the WS-HEU algorithm is not suitable for the distributed setting of web services. This due to the fact that WS-HEU (following the original M-HEU algorithm) starts with a pre-processing step for finding an initial service combination that satisfies all constraints but not necessarily is the best solution, and improves this solution in several rounds of upgrades and downgrades of one of the selected component services. Applying this algorithm in a distributed setting where the QoS data of the different service classes is managed by distributed service brokers would raise very high communication cost among these brokers to find the best composition. The hybrid approach, we propose in this paper solves the composition problem more efficiently and fits well to the distributed environment of web services.

3. SYSTEM MODEL

In our model we assume that we have a universe of web services $S$ which is defined as a union of abstract service classes. Each abstract service class $S_j \in S$ (e.g., flight booking services) is used to describe a set of functionally-equivalent web services (e.g., Lufthansa and Qantas flight booking web services). In this paper we assume that information about service classes is managed by a set of service brokers as described in [Liu et al. 2004; Li et al. 2007]. Web services can join and leave service classes at any time by means of a subscription mechanism.

3.1. Abstract vs. Concrete Composite Services

As shown in Figure 1 we distinguish in the composition process between the following two concepts:

- An abstract composite service, which can be defined as an abstract representation of a composition request $CS_{abstract} = \{S_1, \ldots, S_n\}$. $CS_{abstract}$ refers to the required service classes (e.g.,
flight booking) without referring to any concrete web service (e.g. Lufthansa flight booking web service).

— A concrete composite service, which can be defined as an instantiation of an abstract composite service. This can be obtained by binding each abstract service class in $CS_{abstract}$ to a concrete web service $s_j$, such that $s_j \in S_j$. We use $CS$ to denote a concrete composite service.

### 3.2. QoS Criteria

In our study we consider quantitative non-functional properties of web services, which can be used to describe the quality criteria of a web service [Zeng et al. 2003; Liu et al. 2004]. These can include generic QoS attributes like response time, availability, price, reputation etc, as well as domain-specific QoS attributes like bandwidth for multimedia web services as long as these attributes can be quantified and represented by real numbers. We use the vector $Q_s = \{q_1(s), \ldots, q_r(s)\}$ to represent the $r$ QoS attributes of service $s$, where the function $q_i(s)$ determines the value of the $i$-th quality attribute of $s$. The values of these QoS attributes can be either collected from service providers directly (e.g. price), recorded from previous execution monitoring (e.g. response time) or from user feedbacks (e.g. reputation) [Liu et al. 2004]. The set of QoS attributes can be divided into two subsets: positive and negative QoS attributes. 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 the sake of simplicity, in this paper we consider only negative attributes (positive attributes can be easily transformed into negative attributes by multiplying their values by -1).

### 3.3. QoS Computation of Composite Services

In our previous work [Alrifai and Risse 2009] we focused on sequential compositions. In this work, we extend the QoS computation model to support non-sequential compositions. More specifically, in this study we consider the following four elementary composition constructs as depicted in figure 2, which can be used for building more complex compositions:

1. **Sequential**: a sequence of services $\{s_1, \ldots, s_n\}$ are executed in a strict sequential order one after another.

2. **Loop**: a block of one or more services is executed repeatedly up to a maximum number of $k$ executions. The aggregated QoS values of a loop construct is computed based on the worst case scenario, where the number of iterations equals $k$.

3. **Parallel** (and split/and join): multiple services $\{s_1, \ldots, s_n\}$ are executed concurrently and merged synchronization.

4. **Conditional** (exclusive split/exclusive join): a set of services $\{s_1, \ldots, s_n\}$ are associated with a logical condition, which is evaluated at run-time and based on its outcome one service is executed. The estimated QoS values of a conditional construct are the worst values of the services $\{s_1, \ldots, s_n\}$. For example, the estimated execution price of the conditional construct is computed as the price of the most expensive service among the services $\{s_1, \ldots, s_n\}$.

The QoS vector for a composite service $CS$ with $CS = \{s_1, \ldots, s_n\}$ 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. The value of $q_i'(CS)$ is computed by aggregating the QoS values of the component services $\{s_1, \ldots, s_n\}$. Depending on the QoS attribute and the composition pattern, there can be three different types of aggregation relations: 1) summation, 2) product or 3) minimum/maximum relations. Table I shows examples of such aggregation functions. In this example we consider four different QoS attributes:

— Response time: is the average execution time of the service and is measured by the time between sending a request and receiving a response.

— Price: is the amount of money the requester has to pay for using the service.
— Availability: is the probability that the service is accessible. This is usually measured by the percentage of the service up-time in a given period. The aggregated availability value of a composition is measured by the probability that all composed services are available at execution time, which is usually computed by the product of the individual probabilities.

— Throughput: is the number of requests the service can process per second. The overall throughput of a composition is then determined by the lowest throughput value of the composed services.

In Table I the aggregation function of each of these attributes is shown for each of the four composition constructs mentioned above. Notice that in the conditional construct, only one branch is executed at run-time, which is not known a priori. Therefore, we consider the worst case scenario for estimating the QoS value of the conditional construct. For example, the estimated response time (or price) of a conditional construct that consists of \( n \) branches (such as the one shown in Figure 2) is the maximum response time (or price) among the \( n \) component services, i.e. \( \max_{j=1}^n q(s_j) \). Similarly, for the availability (or throughput) attribute, we use the minimum value among the \( n \) services, i.e. \( \min_{j=1}^n q(s_j) \).

![Composition Constructs](image)

**Fig. 2.** Composition Constructs

<table>
<thead>
<tr>
<th>QoS Attribute</th>
<th>Aggregation Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential</strong></td>
<td></td>
</tr>
<tr>
<td>Response Time</td>
<td>( \sum_{j=1}^n q(s_j) )</td>
</tr>
<tr>
<td>Price</td>
<td>( \sum_{j=1}^n q(s_j) )</td>
</tr>
<tr>
<td>Availability</td>
<td>( \prod_{j=1}^n q(s_j) )</td>
</tr>
<tr>
<td>Throughput</td>
<td>( \min_{j=1}^n q(s_j) )</td>
</tr>
</tbody>
</table>
3.4. Global QoS Constraints

Global QoS constraints represent user’s end-to-end QoS requirements. These can be expressed in terms of upper (and/or lower) bounds for the aggregated values of the different QoS criteria. As mentioned earlier, we only consider negative QoS criteria. Therefore in our model we only have upper bound constraints.

Definition 3.1 (Feasible Selection).
For a given abstract composition \( C_{\text{abstract}} = \{S_1, \ldots, S_n\} \) and a given vector of global QoS constraints \( C' = \{c'_1, \ldots, c'_m\}, 1 \leq m \leq r \) (\( r \) is the size of the QoS vector), 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 \( C_{\text{abstract}} \) and its aggregated QoS values satisfy the global QoS constraints, i.e. \( q_k'(CS) \leq c'_k, \forall k \in [1, m] \).

3.5. Utility Function

In order to evaluate the multi-dimensional quality of a given web service a utility function is used. The function maps the quality vector \( Q_s \) into a single real value, to enable sorting and ranking of service candidates. In this paper we use a Multiple Attribute Decision Making approach for the utility function, i.e. the Simple Additive Weighting (SAW) technique [Yoon and Hwang 1995]. 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 of service candidates. For a composite service \( CS \), the aggregated QoS values are compared with minimum and maximum possible aggregated values. The minimum (or maximum) possible aggregated values can be easily estimated by aggregating the minimum (or maximum) value of each service class in \( CS \). For example, the maximum execution price of \( CS \) can be computed by summing up the execution price of the most expensive service candidate 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 a composition request \( C_{\text{abstract}} = \{S_1, \ldots, S_n\} \) are computed as follows:

\[
Q_{\text{min}}(k) = F_k \prod_{j=1}^{n} (Q_{\text{min}}(j,k))
\]

\[
Q_{\text{max}}(k) = F_k \prod_{j=1}^{n} (Q_{\text{max}}(j,k))
\]

with

\[
Q_{\text{min}}(j,k) = \min_{s \in S_j} q_k(s)
\]

\[
Q_{\text{max}}(j,k) = \max_{s \in S_j} q_k(s)
\]

where \( Q_{\text{min}}(j,k) \) is the minimum value (e.g. minimum price) and \( Q_{\text{max}}(j,k) \) 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 \( F_k \) denotes the aggregation function of the \( k \)-th QoS attribute (see. Table I).

Now the utility of a component web service \( s \in S_j \) is computed as

\[
U(s) = \sum_{k=1}^{r} \frac{Q_{\text{max}}(j,k) - q_k(s)}{Q_{\text{max}}(j,k) - Q_{\text{min}}(j,k)} \cdot w_k
\]
and the overall utility of a composite service is computed as

$$U'(CS) = \sum_{k=1}^{r} \frac{Q_{max}'(k) - q'_k(CS)}{Q_{max}'(k) - Q_{min}'(k)} \cdot w_k$$

(4)

with $w_k \in \mathbb{R}_0^+$ and $\sum_{k=1}^{r} w_k = 1$ being the weight of $q'_k$ to represent user’s priorities.

**Definition 3.2 (Optimal Selection).**

The optimal selection for a given web service composition request $CS_{abstract}$ and a given vector of global QoS constraints $C' = \{c'_1, \ldots, c'_m\}$, $0 \leq m \leq r$, is a feasible selection (according to Definition 3.1) with the maximum overall utility value $U'$.

However, finding the optimal composition requires enumerating all possible combinations of service candidates. For a composition request with $n$ service classes and $l$ service candidate per class, there are $l^n$ possible combinations to be examined. Performing 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.

### 3.6. Problem Statement

The problem of finding the best service composition without enumerating all possible combinations is considered as an optimization problem, in which the overall utility value has to be maximized while satisfying all global constraints. Formally, the optimization problem we are addressing can be stated as follows:

For a given composition request $CS_{abstract} = \{S_1, \ldots, S_n\}$ and a given set of $m$ global QoS constraints $C' = \{c'_1, \ldots, c'_m\}$, find a concrete instantiation $CS = \{s_1, \ldots, s_n\}$ by binding each $S_j$ to a concrete service $s_j \in S_j$ such that:

1. The aggregated QoS satisfy: $q'_k(CS) \leq c'_k, \forall c'_k \in C'$
2. The overall utility $U'(CS)$ is maximized

### 4. THE HYBRID APPROACH

The use of mixed integer programming [Nemhauser and Wolsey 1988] to solve the QoS-aware service composition problem has been proposed by several researchers [Zeng et al. 2003; Zeng et al. 2004; Ardagna and Pernici 2005; 2007]. Binary decision variables are used in the model to represent the service candidates. A service candidate $s_{ij}$ is selected in the optimal composition if its corresponding variable $x_{ji}$ is set to 1 in the solution of the model and discarded otherwise. By re-writing (4) to include the decision variables, the problem of solving the model can be formulated as a maximization problem of the overall utility value given by

$$U'(CS) = \sum_{k=1}^{r} \frac{Q_{max}'(k) - \sum_{j=1}^{n} \sum_{i=1}^{l} q_k(s_{ji}) \cdot x_{ji}}{Q_{max}'(k) - Q_{min}'(k)} \cdot w_k$$

(5)

subject to the global QoS constraints

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

(6)

while satisfying the allocation constraints on the decision variables as

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

(7)

Because the number of variables in this model depends on the number of service candidates (number of variables = $n \cdot l$), this MIP model may not be solved satisfactorily, except for small
instances. Another disadvantage of this approach is that it requires that the QoS data of available web services be imported from the service broker into the MIP model of the service composer, which raises high communication.

To cope with these limitations, we divide the QoS-aware service composition problem into two sub-problems that can be solved more efficiently in two subsequent phases. Figure 3 gives an overview on our approach. In the first phase, the service composer decomposes each global QoS constraint into local constraints on the component services level and sends these constraints to the involved service brokers. In the second phase, each service broker performs local selection to find the best component service that satisfy the local constraints. The two phases of our approach are described in the next subsections in more details.

4.1. Decomposition of Global QoS Constraints

In the first phase of our solution, each QoS global constraint $c'$ is decomposed into a set of $n$ local constraints $\{c_1, \ldots, c_n\}$ ($n$ is the number of abstract service classes in the composite service). The local constraints serve as conservative upper bounds, such that the satisfaction of local constraints guarantees the satisfaction of global constraints. To this end, we use the concept of local quality levels, which are a set of discrete representative values that are extracted from the data of a collection of services. Given the set of local quality levels of each service class, we map each global constraint into one of these local quality levels. We then use the selected quality levels as local thresholds for the corresponding QoS attributes. For example, given a set of candidate web services and their execution prices, we create a list of price levels for that service class (the following subsection describes how levels are determined). The global constraint on total execution price is then mapped to the price levels of service classes.

To avoid discarding any service candidate that might be part of a feasible composition, the decomposition method needs to ensure that the local constraints are not more restrictive than needed. In other words, it is required that the local constraints are relaxed as much as possible while not violating the global constraints. Therefore, we model the QoS constraint decomposition problem as an optimization problem. The goal of this optimization problem is to find a set of local constraints for each service class that cover as many as possible service candidates, while their aggregation does not violate any of the global constraints.

We model this optimization problem as a mixed integer program (MIP) [Nemhauser and Wolsey 1988] and use MIP solving techniques to find the best mapping of global constraints to local quality levels.
levels. Unlike the MIP model in [Zeng et al. 2003; Zeng et al. 2004; Ardagna and Pernici 2005; 2007], our MIP model has much less number of variables (i.e. the representative quality levels instead of actual values of all candidate services) and can be, therefore, solved much faster. In the following we first present our method for selecting local quality levels. After that, we describe how we formulate the QoS constraints decomposition problem as a mixed-integer program.

4.1.1. Selecting Quality Levels. The goal of this step is to determine a small set of discrete QoS values that represent a collection of services. Figure 4 gives an overview of this method. The method takes for each attribute \( q_k \in Q \) as input the QoS values of all \( l \) services in a certain service class \( S_j \), and outputs a set of \( d \) discrete values \( Q_{Ljk} = \{q_1^{jk}, \ldots, q_d^{jk}\} \) such that:

\[
Q_{min}(j,k) \leq q_1^{jk} \leq \ldots \leq q_d^{jk} \leq Q_{max}(j,k).
\]

![Fig. 4. Quality Level Selection- Overview](image)

In this paper we use a simple and effective method for selecting the quality levels, which we describe in Algorithm 1. This algorithm is executed for each QoS attribute \( q_k \in Q \) separately. The first step in this algorithm is to sort the candidate services in class \( S_j \) based on their respective \( q_k \) value. Then the minimum and maximum values are directly added to the set of quality levels. Next, the rest of the sorted set of services is divided into \( d \) subsets. From each subset we randomly select one sample service and use its QoS value as a quality level. The random selection in line 9 of Algorithm 1 excludes services with a QoS value that has been selected in a previous step.

We further explain this method by an illustrating example, which is depicted in Figure 5. In this example, there are 32 candidate services in one service class. The values of a certain QoS attribute \( q \) are shown for each service (e.g. \( q(s_1) = 1 \), \( q(s_{10}) = 4 \) and so on). The goal is to determine \( d \) QoS levels for this QoS attribute that better represent the whole set of services (\( d = 7 \) in this example). The first step is to sort the whole list according to the value of \( q \). Next, the minimum and maximum values are selected directly and the rest of the whole set is divided into 5 equal subsets (e.g. \( s_2 - s_{7}, s_8 - s_{13}, s_{14} - s_{19}, s_{20} - s_{25} \) and \( s_{26} - s_{31} \)). Finally, one service is selected randomly from each subset and its value is used as a QoS level.

Note, that we do not remove duplicate values (i.e. services with the same QoS value). Therefore, the more frequent a given value is, the higher the probability that it is selected as a quality level. This ensures that the selection method takes into account the distribution of the QoS values in the collection. Figure 6 depicts the distribution of the QoS values in this example. The horizontal dashed lines represent the selected QoS level. We observe that most of the selected levels lie in the range \([1, 10]\) (i.e. \( q^1, q^2, q^3 \) and \( q^4 \)), which conforms with the fact that the QoS value of the majority of the services also lie in this range.

As discussed earlier, the purpose of extracting the local quality levels is to use them for decomposing global QoS constraints. The constraint decomposition is performed by a global optimizer, which will decide upon which local quality level should be used as a local constraint. Therefore, we...
can be calculated as i.e. the highest utility value that can be obtained by considering these qualified services. Finally, quality level as a local constraint. The objective of the optimizer will be to select local quality levels that (together) re-assemble the global constraint as well as maximize the aggregated output: of each service candidate in the service class using the utility function (3) and determine $q$ services that would qualify if is stopped and the global optimization method is applied. However, the results of the experimental evaluation, which we conducted on different data sets that include some real-world data set as well as on the user’s constraints. In some scenarios with too many very tight constraints, the decomposition of global constraints into local constraints might fail when using a small number of quality levels. Therefore, finding the optimal value of $d$ depends heavily on the data set as well as on the user’s constraints. In some scenarios with too many very tight constraints, the decomposition of global constraints into local constraints might fail when using a small number of quality levels. Therefore, finding the optimal value of $d$ is a very difficult task. To handle this issue, we propose an iterative method that starts always with a small number of quality levels (like 10 levels) and iteratively duplicates the number of quality levels when needed (i.e. if no feasible decomposition of the global constraints is found). This process continues until a solution is found or $d$ reaches a certain limit. According to our performance analysis (see Section 5.1), the hybrid approach outperforms the global optimization approach as long as $d << \frac{1}{m}$, where $l$ is the (average) number of candidate services in a service class and $m$ is the number of QoS constraints. Therefore, the iterative method is applied as long as the number of QoS levels does not exceed this limit. In the extreme cases, where $d$ reaches the maximum number of $\frac{1}{m}$ before a solution is found, the process is stopped and the global optimization method is applied. However, the results of the experimental evaluation, which we conducted on different data sets that include some real-world data set as well

Algorithm 1 SelectQualityLevels($S_j, q_k, d$)

Input: $S_j$: a set of $l$ candidate services of a certain class, $q_k$: the QoS attribute to be considered, $d$: the required number of quality levels

Output: $QL_{jk}$: a set of $d$ QoS values

1: $QL_{jk} \leftarrow \{}$
2: $S_j \leftarrow \text{sort}(S_j, q_k)$ (sort the services w.r.t. $q_k$)
3: Let $q_{jk}^{\min} \leftarrow S_j[1]$, $q_{jk}^{\max} \leftarrow S_j[l]$
4: $QL_{jk} \leftarrow QL_{jk} \cup \{q_{jk}^{\min}, q_{jk}^{\max}\}$ (add the min and max values to the list of QoS levels)
5: $\text{index} \leftarrow 2$
6: $\text{offset} \leftarrow (l - 2)/d$
7: for $z = 1$ to $d - 2$ do
8: $S_z \leftarrow \{S_i[i \in [\text{index, index + offset - 1}])\}$ (divide the set of services into $d$ subsets)
9: $s_z \leftarrow \text{randomSelection}(S_z)$ (randomly select one service from each subset)
10: $QL_{jk} \leftarrow QL_{jk} \cup \{s_z\}$
11: $\text{index} \leftarrow \text{index} + \text{offset}$
12: end for
13: return $QL_{jk}$

assign each quality level $q_{jk}^*$ a value $p_{jk}^*$ between 0 and 1, which estimates the benefit of using this quality level as a local constraint. The objective of the optimizer will be to select local quality levels that (together) re-assemble the global constraint as well as maximize the aggregated $p$ value (the detailed formulation of the optimization problem and the objective function is described in the next section). This value is determined as follows. First, we compute $h(q_{jk}^*)$, i.e. the number of candidate services that would qualify if $q_{jk}^*$ was used as local constraint. Second, we calculate the utility value of each service candidate in the service class using the utility function (3) and determine $u(q_{jk}^*)$, i.e. the highest utility value that can be obtained by considering these qualified services. Finally, $p_{jk}^*$ can be calculated as

$$p_{jk}^* = \frac{h(q_{jk}^*)}{l} \cdot \frac{u(q_{jk}^*)}{u_{\text{max}}}$$ (8)

where $l$ is the total number of service candidates of service class $S_j$, and $u_{\text{max}}$ is the highest utility value that can be obtained for this class by considering all service candidates.

Intuitively we can infer that the smaller the number of quality levels $d$, the faster the search for a mapping between global constraints and local quality levels will be. However, we can also infer that there is a trade-off between performance and optimality with respect to the selected number of local quality levels. Experimental results, which we present in Section 5 confirm this conclusion. The optimal number of local quality levels to be used (i.e. $d$) is the (average) number of candidate services in a service class and $m$ is the number of QoS constraints. Therefore, the iterative method is applied as long as $d$ does not exceed this limit. In the extreme cases, where $d$ reaches the maximum number of $\frac{1}{m}$ before a solution is found, the process is stopped and the global optimization method is applied. However, the results of the experimental evaluation, which we conducted on different data sets that include some real-world data set as well
Fig. 5. Quality Level Selection - Example

as some synthetic datasets that represent the extreme cases of QoS distributions (correlated, anti-
4.1.2. Constraint Decomposition as an Optimization Problem. Given a global QoS constraint $c_k'$ for a composite service $CS_{abstract} = \{S_1, \ldots, S_n\}$, and a set of $d$ local quality levels of the respective QoS attribute $QL_{jk} = \{q_{1jk}, \ldots, q_{djk}\}$ for each service class $S_j$, the goal of the constraint decomposition is to select an “appropriate” quality level $q_{jk}$ from each service class such that:

1. the aggregation of the selected levels satisfy the global constraint.
2. the number of qualified services is maximized.

We formulate this problem as a mixed-integer program. For the sake of simplicity, we consider here only the sequential composition pattern. In next section, we will show in details how other composition patterns can be handled.

A binary decision variable $x_{zjk}$ is used for each local quality level $q_{zjk}$ such that $x_{zjk} = 1$ if $q_{zjk}$ is selected as a local constraint for the QoS attribute $q_k$ at the service class $S_j$, and $x_{zjk} = 0$ otherwise.

Therefore, we use the following allocation constraints in the model:

$$\forall j, \forall k : \sum_{z=1}^{d} x_{zjk} = 1 \quad 1 \leq j \leq n, 1 \leq k \leq m$$

Note that the total number of variables in the model equals to $n \cdot m \cdot d$, i.e. it is independent of the number of service candidates. If the number of quality levels $d$ satisfies $m \cdot d \leq l$ we can ensure that the size of our MIP model is smaller than the size of the model used in [Zeng et al. 2004; Ardagna and Pernici 2007] (where the number of decision variables is $n \cdot l$), thus can be solved much faster.

In order to ensure that the aggregation of the selected levels satisfy the global constraint, we need to add corresponding constraints into the created MIP model. As the MIP model only supports linear constraints, nonlinear aggregation functions (e.g. multiplication, and minimum functions) need to be transformed into linear constraints.
To this end, we add the following constraint to the model for each QoS attribute that can be aggregated using a summation relation:

$$\sum_{j=1}^{n} \sum_{z=1}^{d} q_{jk} \cdot x_{jk}^{z} \leq c_{k}' \quad , 1 \leq k \leq m$$

(10)

For QoS attributes with a product aggregation function we use the logarithmic function to transform the product relation to a summation relation. We write the constraint as follows:

$$\sum_{j=1}^{n} \sum_{z=1}^{d} \ln(q_{jk}) \cdot x_{jk}^{z} \leq \ln(c_{k}') \quad , 1 \leq k \leq m$$

(11)

For QoS attributes with a minimum aggregation function we add one constraint for each component service:

$$\forall j : \sum_{z=1}^{d} q_{jk}^{z} \cdot x_{jk}^{z} \leq c_{k}' \quad , 1 \leq k \leq m$$

(12)

Similarly, for QoS attributes with a maximum aggregation function we add one constraint for each component service:

$$\forall j : \sum_{z=1}^{d} q_{jk}^{z} \cdot x_{jk}^{z} \geq c_{k}' \quad , 1 \leq k \leq m$$

(13)

The objective function of our MIP model is to maximize the $p$ value (as defined in 8) of the selected local constraints to minimize the number of discarded feasible selections. Therefore, the objective function can be expressed as follows:

$$\text{maximize} \quad \prod_{j=1}^{n} \prod_{k=1}^{m} p_{jk}^{z} \quad , 1 \leq z \leq d$$

(14)

We use the logarithmic function to linearize (14) in order to be able to use it in the MIP model:

$$\text{maximize} \quad \sum_{j=1}^{n} \sum_{k=1}^{m} \sum_{z=1}^{d} \ln(p_{jk}^{z}) \cdot x_{jk}^{z}$$

(15)

By solving this model using any MIP solver, we get a set of local quality levels. These quality levels are then sent to the service brokers to use them as local thresholds when performing local selection.

4.1.3. Handling Complex Composition Models. Recall that the MIP model formulation we described in the previous section assumes a sequential composition model. This assumption was made to simplify the description of the proposed MIP formulation. However, in many practical applications, the composition structure can be very complex involving different types of non-sequential constructs, like for example, conditional branching or multiple parallel execution paths. In order to be able to formulate the MIP model as described in the previous section, we reduce arbitrary composition structures into a sequential one. The reduction is achieved step-by-step. At each step one non-sequential construct is replaced by a single virtual service class. The QoS levels of the virtual service class are aggregated from the QoS levels of service classes it represents, as described below. This procedure continues until no further non-sequential construct exists.

At this point, a sequential representation of the originally non-sequential composition is obtained. The end-to-end QoS constraints are then decomposed into local constraints on the service classes in this sequential composition. The local constraints, which are assigned to a virtual service class serve
as global constraints for the service classes it represents. The constraint decomposition method is then recursively applied on each virtual service class until each service class of the original composition is assigned a set of local constraints.

When replacing a set of service classes in a non-sequential construct by a virtual service class, we are actually replacing a set of random variables in the optimization problem by a single random variable. Therefore, we define the domain of the new random variable (i.e., quality levels) by aggregating the domains of the replaced variables (i.e., quality levels of the replaced service classes).

Figure 7-a shows an example of a parallel composition with two service classes \( S_1 \) and \( S_2 \). \( S' \) is a virtual service representing this composition. The range of the expected response time \( (q_i) \) for each of the classes is shown in Figure 7-b: from \( q_{i1}^{\text{min}} \) to \( q_{i1}^{\text{max}} \) and from \( q_{i2}^{\text{min}} \) to \( q_{i2}^{\text{max}} \) for \( S_1 \) and \( S_2 \), respectively. As this is a parallel composition, the overall response time of \( S' \) is determined by the slowest service. Therefore, the expected minimum response time of \( S' \) is \( q_{i2}^{\text{min}} \) and the expected maximum response time of \( S' \) is \( q_{i1}^{\text{max}} \), which can be computed using the \( \text{max} \) aggregation function according to Table I. To this end, the QoS levels of \( S' \) can be determined by applying Algorithm 1. The input to the algorithm is the set of services from class \( S_1 \) and class \( S_2 \) that have response time in the range \( q_{i2}^{\text{min}} \) to \( q_{i1}^{\text{max}} \).

Another approach for computing the QoS levels of \( S' \) is by exploiting the QoS levels of the composed service classes \( S_1 \) and \( S_2 \). Similar to the computation of the minimum and maximum value of \( q_i \) for \( S' \), the QoS levels of \( S' \) can be computed by aggregating the QoS levels of \( S_1 \) and \( S_2 \). The result is a set of QoS values in the range \( q_{i2}^{\text{min}} \) to \( q_{i1}^{\text{max}} \). Recall that each of the QoS level has a weight \( p \) that is computed according to equation 8 (as described in Section 4.1.1). The objective function of the MIP model is to maximize the \( p \) value for the selected QoS levels. Therefore, we set the \( p \) value of each QoS level of \( S' \) to the lowest value of the corresponding QoS levels of \( S_1 \) and \( S_2 \). In this way, we ensure that by maximizing the \( p \) value of a QoS level of \( S' \) the \( p \) value of the corresponding QoS levels of both \( S_1 \) and \( S_2 \) is maximized. In the following we describe this method in a more formal way.

Given a set of service classes in a composition construct \( S = \{ S_1, ..., S_n \} \), and a set of quality levels \( QL_{ij} = \{ q^1_{kj}, ..., q^d_{kj} \} \) for each \( S_j \in S \) and \( q_k \in Q \), we define the quality levels of the virtual service class \( S* \) that substitutes for \( S \) as follows:

\[
QL_{kS*} = \{ q_{kS*} | q_{kS*} = F_{k,j=1}^{n}(q^z_{kj}) \land p_{kS*} = \min_{j=1}^{n}(p^z_{kj}), 1 \leq z \leq d \}
\]
with the function $F_k$ denoting the aggregation function of the $k$-th QoS attribute. In other words, the $i$-th quality level of $S^*$ is defined by aggregating the $i$-th quality level of each service $s$ in the construct, and its weight $p$ is set to the minimum weight of the aggregated levels.

Consider the example shown in Figure 8. This is an example of a complex composition structure that involves both sequential and parallel executions of services. Table II shows the response time levels of each service class in this example, where the first value in each pair is the value of the response time level $q$ and the second value is the associated weight $p$ according to Equation 8. The complex structure of this composition can be transformed into an equivalent sequential structure in two steps as shown in Figure 8.

![Fig. 8. Transforming Complex Composition Structures into Sequential Structures](image_url)

<table>
<thead>
<tr>
<th>Service</th>
<th>Response Time Levels (msec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(q^1, p^1)$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>(1 msec, 0.10)</td>
</tr>
<tr>
<td>$S_2$</td>
<td>(1 msec, 0.10)</td>
</tr>
<tr>
<td>$S_3$</td>
<td>(1 msec, 0.15)</td>
</tr>
<tr>
<td>$S_4$</td>
<td>(1 msec, 0.05)</td>
</tr>
<tr>
<td>$S_5$</td>
<td>(2 msec, 0.10)</td>
</tr>
<tr>
<td>$S' = \text{sum}(S_2, S_3)$</td>
<td>(2 msec, 0.10)</td>
</tr>
<tr>
<td>$S'' = \text{max}(S', S_4)$</td>
<td>(2 msec, 0.05)</td>
</tr>
</tbody>
</table>
In the first step we replace the sequence \( S_2 \) and \( S_3 \) by a virtual service \( S' \). The QoS levels of \( S' \) are aggregated from those of \( S_2 \) and \( S_3 \) using Formula 16. As \( S' \) represents a sequential construct, the aggregation function \( F \) for response time in this case is the \( \text{sum} \) function (see Table I). The aggregated response time levels of \( S' \) are shown in Table II. Notice that, following Formula 16, the \( p \) value of each QoS level of \( S' \) is equal to the minimum \( p \) value of the corresponding QoS levels of \( S_2 \) and \( S_3 \), i.e., \( p_{s'} = \min(p_{s_2}, p_{s_3}) \).

In the second step we replace the parallel construct involving \( S' \) and \( S_4 \) by a virtual service \( S'' \). Similarly, the QoS levels of \( S'' \) are aggregated from the QoS levels of \( S' \) and \( S_4 \) according to Formula 16. As \( S'' \) represents a parallel construct, its response time levels are aggregated using the \( \text{max} \) function (see Table I). These values are also shown in Table II.

The resultant structure is a sequence of services \( S_1, S'' \) and \( S_5 \). Applying the MIP formulation steps from previous section we can decompose global QoS constraints into local constraints for \( S_1, S'' \) and \( S_5 \). The local constraints of \( S'' \) are then used as global constraints for the services \( S' \) and \( S_4 \). By further decomposing these constraints we obtain local constraints for \( S' \) and \( S_4 \). Finally, by decomposing the constraints of \( S' \) we obtain local constraints for \( S_2 \) and \( S_3 \).

We explain this with the following numerical example. Assume that the end-to-end response time constraint for this composition is 10 msec. By decomposing this global constraint into local constraints on \( S_1, S'' \) and \( S_5 \), we get the following local constraints, respectively: 1 msec, 7 msec and 2 msec. This results in from mapping the global constraint to the following local constraints, respectively: (1, 0.10) , (7, 0.55) and (2, 0.10). Notice, that this mapping is the only mapping that satisfy the global constraint, while maximizing the \( p \) value.

Next, the local constraint of \( S'' \) (i.e. 7 msec) is decomposed into local constraints on \( S' \) and \( S_4 \). Since the aggregation type of the parallel construct is the \( \text{max} \) operator, we can directly apply the global constraint value (7 msec) as a local constraint for \( S' \) and \( S_4 \). For \( S_4 \) this maps to the local QoS level (7, 0.75).

Finally, the local constraint of \( S' \) (i.e. 7 msec) is decomposed into local constraints on \( S_2 \) and \( S_3 \). Since the aggregation type of the sequential construct is a \( \text{sum} \) relation, we need to apply the constraint decomposition method to map the global constraint into local QoS levels. This results in mapping the global constraint to the following QoS levels of \( S_2 \) and \( S_3 \), respectively: (3, 0.60) and (4, 0.55).

Hence, the end-to-end response time constraint, 10 msec, was decomposed into the following local constraints, 1, 3, 4, 7 and 2 msec for \( S_1, S_2, S_3, S_4 \) and \( S_5 \), respectively.

### 4.2. Local Selection

After decomposing global QoS constraints into local ones, the second step of our solution is to perform local selection for each service class independently. Upon the receipt of local constraints and user' preferences from the service composer, each service broker performs the local selection and returns the best web service candidate to the service composer. The received local constraints are used as upper bounds for the QoS values of component services. Web services that violate these upper bounds are skipped from the selection. A list of qualified services is created and sorted by their utility values.

The use of (3) for this purpose is not appropriate for the following reason. This utility function compares the distance \( Q_{\text{max}}(j, k) - q_k(s_{ji}) \) between the quality value of a service candidate \( s_{ji} \) and the local maximum value in its class \( S_j \) with the distance \( Q_{\text{max}}(j, k) - Q_{\text{min}}(j, k) \) between the local minimum and maximum values. This scaling approach can be biased by local properties leading to local optima instead of global optima. Therefore, we compare the distance \( Q_{\text{max}}(j, k) - q_k(s_{ji}) \) with the distance between the maximum and minimum overall quality values \( Q_{\text{max}}(k) - Q_{\text{min}}'(k) \). This scaling method ensures that the evaluation of service candidates is globally valid, which is important for guiding local selection in order to avoid local optima. The scaling process is then followed by a weighting process for representing user’s over the different QoS attributes. We
compute the utility $U(s_{ji})$ of the $i$-th service candidate in class $S_j$ as

$$U(s_{ji}) = \sum_{k=1}^{r} \frac{Q_{\text{max}}(j,k) - q_k(s_{ji})}{Q_{\text{max}}(k) - Q_{\text{min}}(k)} \cdot w_k$$

with $w_k \in \mathbb{R}_0^+$ and $\sum_{k=1}^{r} w_k = 1$ being the weight of $q_k$ to represent user’s priorities.

5. PERFORMANCE STUDY

The aim of this evaluation is to validate our hypothesis that our approach achieves close-to-optimal results with a much lower computation time compared to “pure” global optimization approach as proposed by [Liu et al. 2004; Zeng et al. 2004; Ardagna and Pernici 2007]. In the following we use the label “hybrid” to refer to our solution and the label “global” to refer to the “pure” global optimization approach.

5.1. Performance Analysis

The scalability of QoS-based service composition systems is affected by the time complexity of the applied algorithm. There are three factors that determine the size of the composition problem: the number of required service classes $n$, the number of service candidates per class $l$, which we assume to be equal for all classes, and the number of global QoS constraints $m$. As the problem can be modeled as a Multi-dimensional Multiple-choice Knapsack Problem (MMKP), which is known to be NP-hard [Pisinger 1995], the time complexity of any exact solution is expected to be exponential. Existing global optimization solutions model the service selection problem as a standard mixed integer program (MIP). The worst case time complexity of MIP solvers using the Simplex method and Branch and Bound algorithms is an exponential function with respect to the problem size (i.e. $n$, $l$ and $m$) [Maros 2003]. Therefore, MIP based solutions are only applicable for small size composition problems, where the number of service candidates $l$ and the number of involved classes $n$ is very limited.

In our Hybrid approach, we use mixed integer programming to solve part of the problem, namely, the decomposition of the global QoS constraints into local ones. The actual selection of services, however, is done using distributed local selection strategy, which is very efficient and scalable. The local utility computation for service candidates has a linear complexity with respect to the number of service candidates, i.e. $O(l)$. As service brokers can perform the local selection in parallel, the total time complexity of this step is not affected by the number of service classes, hence, the complexity of the second step remains $O(l)$.

The time complexity of our approach is dominated by the time complexity of the constraint decomposition part. The number of decision variables in our MIP model is $n \cdot m \cdot d$, where $n$ is the number of service classes, $m$ is the number of global QoS constraints and $d$ is the number of quality levels. Consequently, the time complexity of our approach is independent on the number of available web services, which makes it more scalable than existing solutions that rely on “pure” global optimization. By selecting a low number of quality levels $d$ with $1 < d \ll \frac{L}{m}$ we ensure that the size of the MIP is much smaller compared to the MIP model used in the global optimization approaches in [Liu et al. 2004; Zeng et al. 2004; Ardagna and Pernici 2007].

5.2. Experimental Evaluation

We have conducted extensive simulations to evaluate the performance of the proposed QoS-aware service selection approach, which we describe in this section.

5.2.1. The Dataset. In our evaluation we experimented with two types of datasets: real and synthetic datasets. The first is the publicly available dataset QWS\(^1\), which comprises measurements of 9 QoS attributes for 2500 real-world web services. These services were collected from public

\(^1\)http://www.uoguelph.ca/qmahmoud/qws/index.html/
sources on the Web, including UDDI registries, search engines and service portals, and their QoS values were measured using commercial benchmark tools. Table III lists the QoS attributes in this dataset and gives a brief description of each attribute. More details about this dataset can be found in [Al-Masri and Mahmoud 2008].

In order to make sure that the results of our experiments are not biased by the used QWS dataset, we also experimented with three synthetically generated datasets with larger number of services and different distributions. For this purpose, we used a publicly available synthetic generator\footnote{http://randdataset.projects.postgresql.org/} 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 (iQoS), 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.

5.2.2. Evaluation Methodology. For the purpose of our evaluation, we considered a scenario, where a composite application comprises services from $n$ different service classes ($n$ varies in our experiments between 10 and 50 classes). Thus, we randomly partitioned each of the aforementioned datasets into $n$ service classes. We consider both sequential and arbitrary complex composition structures. More details about the construction of complex compositions are given later in the following subsections.

We then created several vectors of up to 9 random values to represent the users end-to-end QoS constraints. Each constraints vector corresponds to one composition request, for which one concrete service needs to be selected from each class, such that all end-to-end constraints are satisfied, while the overall utility value is maximized. We solved each composition request using the following methods:

— **Global**: this is the global optimization method [Liu et al. 2004; Ardagna and Pernici 2007] with all service candidates represented in the MIP model. This method returns the optimal selection of services, and therefore is used as a baseline in our experiments.

— **Hybrid**: this is our proposed method in this paper for combining global optimization with local selection, based on the concept of constraint decomposition.

\begin{table}[h]
\centering
\begin{tabular}{|c|p{6cm}|c|}
\hline
QoS Attribute & Description & Units Of Measurement \\
\hline
Response Time & Time taken to send a request and receive a response & millisecond \\
\hline
Availability & the probability that the service is accessible & percent \\
\hline
Throughput & Total number of invocations for a given period of time & invocations / second \\
\hline
Likelihood of success & Number of response/number of request messages & percent \\
\hline
Reliability & Ratio of the number of error messages to total messages & percent \\
\hline
Compliance & To which extent a WSDL document follows the WSDL spec. & percent \\
\hline
Best Practices & To which extent a web service follows the Web Services Interoperability (WS-I) Basic Profile & percent \\
\hline
Latency & Time the server takes to process a given request & millisecond \\
\hline
Documentation & Measure of documentation (i.e. description tags) in WSDL & percent \\
\hline
\end{tabular}
\caption{QoS attributes in the QWS dataset}
\end{table}
A Hybrid Approach for Efficient WS Composition with QoS Constraints

WS-HEU: this is the heuristic method proposed in [Yu et al. 2007], which is a modification of the original heuristic M-HEU [Akbar et al. 2001] for solving Multi-dimensional Multiple-choice Knapsack Problems (MMKP).

We then recorded the required computation time (average of 50 executions) by each of the aforementioned methods to return the selection of services. As our hybrid solution is an approximate solution, we have evaluated the quality of the results obtained by our solution by comparing it with the optimal results obtained by the global optimization approach. We compute the optimality of the results of the hybrid approach by comparing the overall utility value ($u_{hybrid}$) of the selected services to the overall utility value ($u_{global}$) of the optimal selection obtained by the global approach, i.e.:

$$optimality = \frac{u_{hybrid}}{u_{global}}$$

We used the open source (Mixed Integer Programming) LpSolve system lpsolve version 5.5 [Michel Berkelaar and Notebaert] for solving the MIP model in both approaches. The experiments were conducted on a HP ProLiant DL380 G3 machine with 2 Intel Xeon 2.80GHz processors and 6 GB RAM. The machine is running under Linux (CentOS release 5) and Java 1.6.

5.2.3. Effect of Number of Quality Levels. In the first experiment we evaluated the effect of the chosen number of quality levels $d$ in the hybrid approach. For this purpose we created a test case of a composition request of 5 service classes with 500 candidate services per class. We then randomly created a vector of 7 end-to-end QoS constraints. We solved the selection problem using both the Global approach and the Hybrid approach while in the Hybrid approach we solved the problem several times, each time with a different number of chosen quality levels: i.e. 10, 20, 30, 40 and 50 quality levels. The results of this experiment are shown in Figure 9. To the left side of this figure a performance comparison of the two methods is shown. We notice that the Hybrid method is much faster than the Global method, while still able to achieve a very close-to-optimal results as shown on the second chart to the right side of Figure 9. We also notice that using more quality levels improves the optimality if the selection, but on the other hand, it also imposes more overhead in terms of computation time.

Hence, there is a trade-off between performance and optimality with respect to the chosen number of quality levels. Moreover, in some cases with very constrained composition requests, the decomposition of global constraints into local constraints might fail when using a small number of quality levels. To handle this issue, we apply the iterative method, which we described in Section 4.1.1. The iterative method starts always with a small number of quality levels (10 levels in this case) and if no feasible decomposition of the global constraints is found, the number of QoS levels is iteratively duplicated. The method continues until a solution is found. This approach is applied in the following experiments, and the practice shows that in almost all cases, with all different types of datasets, a solution is found after a few number of iterations (up to 3 iterations, i.e the number of levels used are 10, 20 or 40 levels).

5.2.4. Performance Comparison for Sequential Compositions. In the following we present a performance comparison of the different approaches with respect to the Number of Candidate Services, Number of Service Classes and Number of QoS constraints for sequential compositions.

Performance vs. Number of Candidate Services: In Figure 10 we compare the performance of our hybrid approach with the performance of the Global and WS-HEU approaches with respect to the number of service candidates. The graphs show the measured average computation time of each of the three selection methods. The number of service candidates per class $l$ varies from 100 to 1000 services per class, while the number of service classes $n$ is set to 10 classes and the number of QoS constraints to 5 end-to-end constraints.

Although slightly slower than WS-HEU with the correlated dataset, our Hybrid approach remains far faster than the Global approach in all datasets and significantly faster than WS-HEU in the
other three datasets: QWS, iQoS and aQoS. By increasing the number of service candidates, the required computation time of the Hybrid approach increases very slowly, which makes our solution more scalable.

— *Performance vs. Number of Service Classes*: In the experiment shown in Figure 11 we study the performance of both approaches with respect to the number of service classes \( n \) in the composition. The number of service classes varies in this experiment from 10 to 50 classes, while the number of service candidates per class \( l \) is fixed to 500 and the number of constraints to 5 constraints. The results of this experiment indicate that the performance of all three methods degrade as the number of service classes increases. However, the Hybrid approach still outperforms...
the Global approach in all datasets. Again, we observe that the Hybrid approach outperforms the WS-HEU approach in all datasets except the correlated-dataset, where it performs slightly slower.

![Performance vs. Number of Service Classes](image)

--- *Performance vs. Number of QoS Constraints:* In the next experiment we evaluate the performance of the selection methods against the number of end-to-end QoS constraints. For this purpose, we set the number of classes $n$ to 10, the number of candidate services $l$ to 500 services per class, and vary the number of constraints between 1 and 9 constraints (recall that the number of QoS attributes in our datasets is 9). The results of this experiment are shown in Figure 12. We observe that the computation time of the Hybrid approach increases quickly as the number of constraints increases. However, we also observe that even in the worst case with maximum number of QoS constraints (i.e. 9 constraints) the computation time of the Hybrid approach remains far less than the computation time required by the Global approach. On the other hand, the performance of the WS-HEU approach differs from a dataset to another. We notice that in general the computation time required by this method decreases as the number of constraints increases, although still higher than the computation time of the Hybrid approach in most of the cases. The reason for this behavior is that the WS-HEU method relies on a pre-processing step in which most of the infeasible selections are filtered out before the real search for the optimal solution starts. Consequently, as the number of constraints increases, the number of infeasible selections that are filtered out in his early stage increases, which in turn reduces the search space and hence the required time for finding the best solution.

5.2.5. Optimality Comparison for Sequential Compositions. Next we present the evaluation of the quality of the results obtained by our solution in terms of the achieved optimality. As we discussed
earlier in this section, the optimality of an approximate solution (i.e. the returned selection of services) is measured by comparing its utility value with the utility value of the optimal solution obtained by the Global method. We also measured the optimality of the selection obtained by the WS-HEU method and compare it with the optimality of our Hybrid approach.

— Optimality vs. Number of Candidate Services: Figure 13 shows the achieved optimality with different datasets and a varying number of candidate services. The results indicate that the Hybrid approach was able to achieve very close-to-optimal results. The results also show that the WS-HEU approach was able to achieve even higher optimality than the Hybrid approach.
— **Optimality vs. Number of Service Classes:** Figure 14 shows the achieved optimality with different datasets and a varying number of service classes. The results indicate that the Hybrid approach was able to achieve very close-to-optimal results in all cases (above 98% in average). The results also show that the WS-HEU approach was able to achieve even higher optimality than the Hybrid approach.

— **Optimality vs. Number of QoS Constraints:** Figure 15 shows the achieved optimality with different datasets and a varying number of end-to-end QoS constraints. The results indicate that the Hybrid approach was able to achieve very close-to-optimal results in all cases, although with the anti-correlated dataset the optimality decreases slowly as the number of constraints increases. The achieved optimality by the WS-HEU method on the other hand remains unaffected by the number of constraints in all datasets.

5.2.6. **Communication Cost Comparison for Sequential Compositions.** The purpose of this experiment is to investigate the communication overhead of deploying the selection methods (i.e. Hybrid and WS-HEU) in a distributed setting. We measure the overhead in terms of the number of messages that need to be exchanged between the composer and the distributed service brokers. Without loss of generality, in this experiment we assume that each service class is managed by a separate service broker. The results of this experiment proves our hypothesis that the WS-HEU method is not suitable for distributed environments. This is due to the fact that WS-HEU optimizes the service selection by undertaking several iterations of downgrading and upgrading of local selections (i.e.
replacements of already selected services) until no further optimization is possible. This process requires extensive communication with the respective service brokers in each round, which in a distributed environment can lead to a high number of exchanged messages. In the results shown in Figure 16 we see that while the number of exchanged messages in the Hybrid approach (for obtaining local quality levels) remains very limited, the number of exchanged messages in the WS-HEU method is much higher and is increasing almost constantly.

![Communication Cost vs. Number of Service Classes](image)

**Fig. 16.** Communication Cost vs. Number of Service Classes

5.2.7. **Evaluation with Complex Composition Structures.** The computation model of the hybrid approach assumes a sequential structure. Therefore, in Section 4.1.3, we described how the hybrid approach can be used for handling arbitrary complex compositions. The non-sequential constructs are transformed into sequential ones before the algorithm for decomposing QoS constraints is applied. For some nested composition structures, it is required that the transformation/decomposition steps are repeated for each of the nested levels. For a composition with \(n\) nested levels the decomposition of constraints has to be performed \(n\) times.

In this experiment we evaluate the scalability of the hybrid approach against the level of complexity of service compositions, expressed by the number of nested levels.

For this purpose, we start with a sequential composition of 5 service classes (i.e. nesting level = 0) as shown in Figure 17-a. We then incrementally, increase the nesting level by replacing one service with a parallel construct in each step. In the example shown in Figure 17-b, service \(S_3\) is replaced with a parallel construct including \(S_6, S_7\) and \(S_8\). In order to increase the nesting level further, \(S_8\) is replaced with \(S_9, S_{10}\) and \(S_{11}\) in Figure 17-c. In the experiment we increased the nesting up to level 30.

We used the QWS dataset for this experiment. The dataset was randomly partitioned into a set of groups of services. Each group of services represents one service class. Similar to the evaluation methodology described in Section 5.2.2, a set of random end-to-end QoS constraints are generated and each of the service selection methods is applied to find a good selection of services that satisfy these constraints and maximize the overall utility value.

Before we present the results of this experiment, we explain how the Global approach and the WS-HEU algorithm were applied to solve the selection problem for complex compositions.

As the Global approach cannot handle complex compositions straightforwardly, we followed an approach similar to the one described in [Zeng et al. 2004]. We identify all sequential execution paths in the composition. A sequential path is a path from the first service in the composition to the last service without any branching. There are three sequential paths in the example shown in Figure 17-c: 1) a sequential path including \(S_1, S_2, S_6, S_7, S_4\) and \(S_5\), 2) a sequential path including \(S_1, S_2, S_9, S_{10}, S_4\) and \(S_5\), and 3) a sequential path including \(S_1, S_2, S_{11}, S_4\) and \(S_5\). All the
identified paths must satisfy the end-to-end QoS constraints. Therefore, for each QoS constraint, one constraint is added to MIP model for each of the sequential paths.

For the WS-HEU algorithm, whenever the algorithm checks the feasibility of a certain selection, it also verifies that all sequential paths satisfy the end-to-end constraints.

The results are shown in Figure 18 and Figure 19. The performance comparison in Figure 18 has shown that the computation time increases as the level of nesting increases, which is an expected behavior. The global approach, requires more computation time because the number of constraints in the MIP model increases as the number of sequential paths increases. Similarly, the WS-HEU algorithm also requires more computation time as the feasibility check requires more time with more sequential path. In the Hybrid approach the number of required constraint decompositions increases with the increase of nesting levels. However, the results of this experiment has shown that the Hybrid approach is more scalable than the other two methods as the computation time increases much slower than in the case of the Global and WS-HEU methods. Moreover, Figure 19 shows that both the Hybrid and the WS-HEU methods are still able to achieve close-to-optimal results for higher level of nesting in complex compositions.

The reason for this behavior is the difference in the design of the three approaches. In the case of the Global and WS-HEU approaches, increasing the nesting level implies increasing the number of services and the number of constraints, which results in increasing the search space, and hence, the time required to find a feasible solution that satisfies all constraints, while maximizing the overall utility. On the other hand, the performance of the Hybrid approach is less sensitive to the number of services (as discussed earlier in Sections 4.1.2 and 5.1), therefore, increasing the nesting level does not imply increasing the size of the problem, instead, it requires applying the method repeatedly until a set of local constraints for each individual service class have been determined. As the overhead of applying the Hybrid approach is inherently very low compared to the Global and WS-HEU approaches (see results in Figure 10 and Figure 11), the overall overhead of the repeated execution of the Hybrid approach remains far below the overhead of applying these methods on a nested composition.
5.2.8. Summary of the Results. The results of the experimental evaluation have shown that the Hybrid approach significantly outperforms the Global approach in terms of computation time, while being able to achieve a close-to-optimal results. The results have also shown that the Hybrid approach is more scalable than the WS-HEU approach with respect to the number of candidate services and service classes. However, the WS-HEU approach scales better than the Hybrid approach with respect to the number of QoS constraints. On the other hand, our Hybrid approach imposes much less communication overhead in comparison with WS-HEU when applied in a fully distributed setting. The results of the experimental evaluation have also shown that the Hybrid approach scales much better than the Global and WS-HEU approaches with respect to the complexity of the composition structure.

6. CONCLUSION AND FUTURE WORK
In this paper we presented an efficient heuristic for solving the QoS-based service composition problem, which is known to be NP-hard. We combine global optimization with local selection methods to benefit from the advantages of both worlds. The proposed hybrid method allows to dramatically reduce the overhead of finding a combination of services that satisfy a set of end-to-end QoS constraints compared to existing global solutions. Our evaluations show a significant improvement in
terms of computation time, while achieving close to optimal results. This is especially useful for applications with dynamic changes and real-time requirements.

Currently we are working on an intelligent method for determining quality levels using clustering algorithms. This can help improving the scalability of the hybrid approach against the number of QoS constraints. We are also investigating the applicability of a skyline model for reducing the search space by pruning dominated services and focusing on highly qualified ones. This can have a great impact on the scalability of the selection method and enable sorting and ranking candidate services. A protocol for coordinating the distributed service brokers, which are involved in a QoS optimization process, is also part of our future work.

REFERENCES


Received January 2011; revised June 2011; accepted August 2011