SOCA (2015) 9:107–138
DOI 10.1007/s11761-014-0154-x

ORIGINAL RESEARCH PAPER

An approach for selecting best available services through a new method of decomposing QoS constraints

Chellammal Surianarayanan · Gopinath Ganapathy · Manikandan Sethunarayanan Ramasamy

Received: 19 March 2013 / Revised: 29 December 2013 / Accepted: 26 January 2014 / Published online: 20 February 2014

Abstract Quality of Service (QoS) plays as a discriminating factor for selecting appropriate services that meet the given user's non-functional requirements during service composition. There is a compelling need to select suitable services quickly so that the composition can meet dynamic needs. Recently, local selection approaches for QoS-based selection have been put forward toward reduced time complexity. A methodology for selecting the best available service combination for a given user requirement (workflow) with a new method of decomposing QoS constraints is proposed in this paper. The methodology consists of two phases, namely 'Constraint Decomposition Phase' and 'Service Selection Phase'. In the Constraint Decomposition Phase, a unique method is proposed to decompose the given non-functional (global or workflow level) constraints into local constraints for individual tasks in the workflow. Each individual task with its local constraints forms a subproblem. In the Service Selection phase, each subproblem is resolved by finding the best available service from its respective service class using an iterative searching procedure. A prototype has been implemented, and the low computation time of the proposed method makes it well suited to dynamic composition. The proposed method of decomposing constraints is independent of number of services in a service class, and the method is applicable to any combinational workflow with AND, OR and Loop patterns. Further, a new method for computing response time of OR execution pattern which guarantees successful execution of each path in an OR pattern is a remarkable contribution of this work.

Keywords Local service selection · QoS-based service selection · Decomposition of QoS constraints · Web services · Service composition

Electronic supplementary material The online version of this article (doi:10.1007/s11761-014-0154-x) contains supplementary material, which is available to authorized users.

C. Surianarayanan (✉) · G. Ganapathy
School of Computer Science and Engineering,
Bharathidasan University, Tiruchirappalli 620024, India
e-mail: chelsrsd@rediffmail.com

G. Ganapathy
e-mail: gganapathy@gmail.com

M. S. Ramasamy
Department of Mathematics, Bharathidasan University Constituent College, Lalgudi 621601, India
e-mail: manirs2004@yahoo.co.in

1 Introduction

Service Oriented Architecture (SOA) promotes the development of complex business applications through *service composition* where more than one services are combined according to a specific pattern in order to achieve the given requirement. The business requirements are usually represented as workflow. A workflow consists of a combination of tasks. Each task represents an abstract function having input and output parameters. The tasks are implemented and published as services by different service providers. During service composition, services that implement various tasks of the workflow are discovered and combined as a composite service. Each service has a set of QoS attributes referring to non-functional attributes such as availability, latency, scalability, cost, response time and reliability. For each task, alternative services having same function but different QoS are growing in large number and the group of functionally similar services would form the 'service class' [1] for that task.

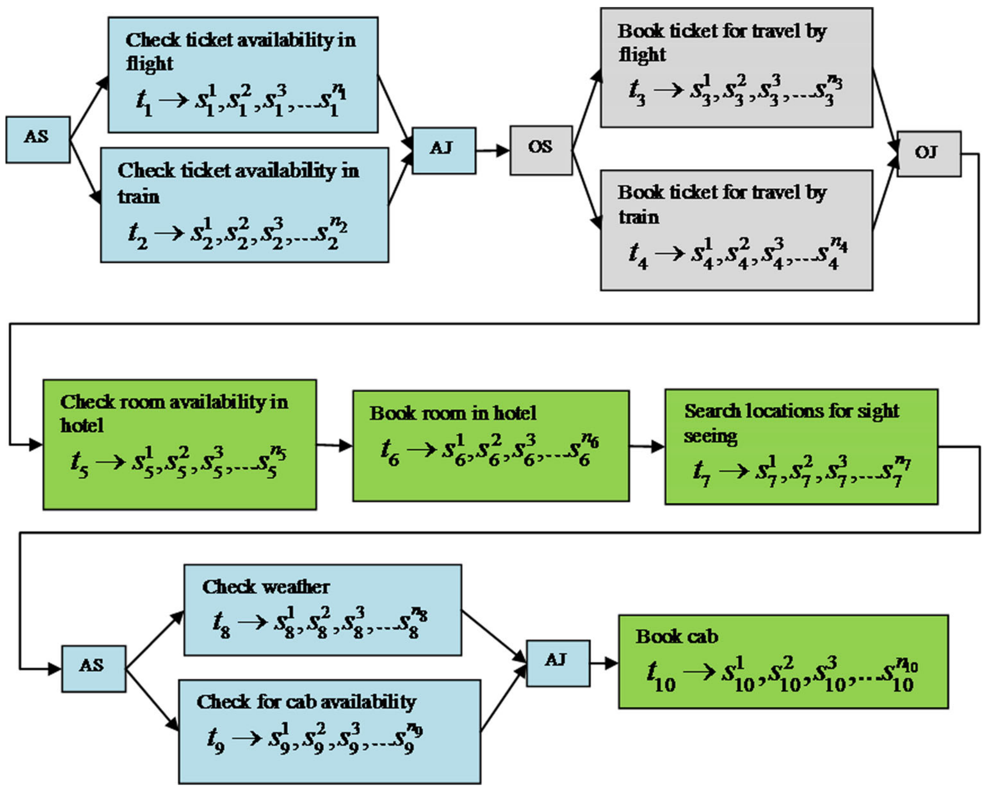
AS
Check ticket availability in flight
$t_1 \rightarrow s_1^1, s_1^2, s_1^3, \ldots s_1^{n_1}$
Check ticket availability in train
$t_2 \rightarrow s_2^1, s_2^2, s_2^3, \ldots s_2^{n_2}$
AJ
OS
Book ticket for travel by flight
$t_3 \rightarrow s_3^1, s_3^2, s_3^3, \ldots s_3^{n_3}$
Book ticket for travel by train
$t_4 \rightarrow s_4^1, s_4^2, s_4^3, \ldots s_4^{n_4}$
OJ
Check room availability in hotel
$t_5 \rightarrow s_5^1, s_5^2, s_5^3, \ldots s_5^{n_5}$
Book room in hotel
$t_6 \rightarrow s_6^1, s_6^2, s_6^3, \ldots s_6^{n_6}$
Search locations for sight seeing
$t_7 \rightarrow s_7^1, s_7^2, s_7^3, \ldots s_7^{n_7}$
AS
Check weather
$t_8 \rightarrow s_8^1, s_8^2, s_8^3, \ldots s_8^{n_8}$
Check for cab availability
$t_9 \rightarrow s_9^1, s_9^2, s_9^3, \ldots s_9^{n_9}$
AJ
Book cab
$t_{10} \rightarrow s_{10}^1, s_{10}^2, s_{10}^3, \ldots s_{10}^{n_{10}}$

Fig. 1 A typical workflow of travel plan

The process of selecting appropriate services for composition must be based on QoS characteristics of services as QoS is part of customer's demands as in the example query *Find Currency conversion (currency conversion is the functional demand) service with cost of the service* <\$50 *(cost constraint is QoS/non-functional demand)*. QoS requirements of clients are based on the nature of applications, and in certain applications, the QoS demands are very crucial that the QoS demands are dominating in deciding the success of an application. This is illustrated with the following example.

Consider a client who records the weather conditions at a location during cyclone with an intention of modeling atmospheric dispersion characteristics under cyclonic conditions. Here, the client looks for a *Weather Service with response time of the service strictly less than one second*. Let us consider another client who looks for *Weather Service with cost of the service less than* \$50 with an intention of simply knowing the existing weather conditions at a given location. In the case of first client, response time of the service is important; the client may pay even higher cost to get the desired response time, whereas in the case of second client cost constraint is important and this client may be satisfied with a service having poor response time of say, even one minute. In case, if the first client happens to use a service which does not have the required response time (in this example, less than one second), his purpose itself will not be met and his application will be unsuccessful. Similarly, in the case of second client if cost is higher than \$50, the client may not even avail the service. Hence, services should be selected for composition according to QoS demands of clients.

Further, in typical service composition, services from different domains will be participating in different execution patterns, such as AND, OR and sequential, to achieve a given requirement. For example, consider a typical workflow of *Travel Plan* given in Fig. 1. In Fig. 1, 'AS' denotes AND Split, 'AJ' denotes AND Join, 'OS' denotes OR Split and 'OJ' denotes OR Join. The given Travel Plan workflow consists of 10 tasks, namely $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9$ and t_{10} which belong to different organizations, namely Flight, Train, Hotel, Cab and Weather. The tasks are combined in different patterns, such as AND, OR and sequential. For each task, several services having similar functionality but varying QoS characteristics would form its service class. For example, services, namely $s_1^1, s_1^2, s_1^3, \ldots s_1^{n1}$ are available to implement t_1. Services, namely $s_2^1, s_2^2, s_2^3, \ldots s_2^{n2}$ are available to implement t_2. Similarly, for each task in Fig. 1, several functionally similar services with varying QoS characteristics are available to implement that task. Despite the complexity of workflow, the runtime performance of service composition is crucial for distributed applications and applications having online customers [16,29].

Table 1 Number of service combinations and time taken by global selection with respect to number of services

Number of services per service class	Global approach	
	Number of service combinations	Time taken in minutes
1	1	0.000001
2	1,024	0.000006
3	59,049	0.000018
4	1,048,576	0.000517
5	9,765,625	0.004683
6	60,466,176	0.0299
7	$2.82e^{+08}$	0.1352
8	$1.07e^{+09}$	0.51975
9	$3.49e^{+09}$	1.658033
10	$1e^{10}$	4.806633
11	$2.59e^{+10}$	12.30817
12	$6.19e^{+10}$	29.65435
13	$1.38e^{+11}$	418.788

Hence, in this real scenario of various tasks from different domains, with each task having several functionally similar services, the problem of service selection is to find appropriate service combination which best satisfies the clients according to their QoS requirements within relatively short time interval as expected by clients.

To figure out service selection methods, there are two major categories, namely global and local [30]. Global approach performs service selection at composite service level. In this approach, one service from each service class is selected to form a composite service. Then, the values of QoS attributes of composite service are computed and tested for fulfillment against user's QoS constraints. The possible number of service combinations to be searched to find an appropriate composition of 'n' service classes with 'l' number of services in each service class is l^n.

In the given example scenario, to illustrate how the number of combinations and time taken for finding optimal solution using global approach increase with respect to number of services, the number of services per task is increased from 1 to 13. The number of service combinations and time taken for selecting services for different number of services are given in Table 1.

From Table 1, the number of combinations is found to increase exponentially with number of services. Hence, the time taken for finding suitable service combination also increases exponentially. This illustrates how global approaches such as Linear Programming (LP) or Mixed Integer Linear Programming (MILP) [6,7,17,30], heuristics [5,29] become impractical when number of services grows. Though the other methods such as genetic algorithm [8], Ant Colony Optimization [31] and hybrid methods [11,27] focus on reducing the computation time, they are applicable only to service composition with limited number of services. A broker-based framework for QoS-aware service composition with end-to-end constraints presented in [28] gives less attention to time complexity issue.

Nowadays, local selection methods are widely used to reduce the time complexity of service selection. Local selection methods achieve time reduction by dividing the problem of selecting service combination for a given workflow into a set of independent task level subproblems. Local selection method treats selecting services for the workflow as main problem and selecting service for individual task in the workflow as subproblem. In this approach, the possible number of service combinations of 'n' service classes with 'l' number of services in each service class is only $l \times n$.

Although the existing local selection methods such as [2,3,21] are efficient in meeting global QoS requirements, they can handle only sequential workflows, whereas common business workflows contain execution patterns such as AND, OR and Loop. Also, the methods such as [2,3] use Mixed Integer Programming (MIP) to find local constraints, which has poor time characteristics when number of services grows. Another weakness that has to be resolved is with respect to the computation of values of QoS attributes for an OR pattern. Though the methods [9,15,19,22] provide a formal way (computation of expected values of QoS attributes) to compute QoS attributes of an OR execution pattern, there is no guarantee that the execution of all alternate paths of the OR pattern will succeed when they are given a chance for execution. Alternately, [12] formulates the local service selection as LP model for finding optimal services. Theoretically, an LP model takes a user-defined utility function, a set of decision variables and a set of linear constraints as inputs. The model optimizes the utility function by optimizing the decision variables subject to the given constraints. Ultimately, the model produces optimal values for decision variables and utility function. Though the LP model computes optimal values for decision variables and utility function, it never helps in finding the identity of optimal service (which is prerequisite for composition). Further, though we can find the optimal values using LP models, it is not mandatory that a service with optimal value should be available in a service repository. Therefore, an efficient local selection method that alleviates the above shortcomings should be designed.

In this paper, local selection based methodology is proposed in order to alleviate the above issues. QoS requirements consist of (i) a set of QoS constraints and (ii) a set of QoS preferences. QoS constraints are based on range of QoS attributes, whereas QoS preferences are based on user's priorities over different QoS attributes. QoS constraints represent upper and/or lower bound values that QoS attributes can take.

For example, *Find a Weather Service with QoS constraints: Cost <=\$500 and response time <=* 1 s. QoS preferences refer to priorities or preferences of users over different QoS attributes. For example, while querying, clients may specify their QoS preferences as *80 % priority to response time and 20 % priority to cost*. QoS constraints and preferences vary from application to application.

To reduce the complexity of selecting the best available service combination, a set of independent subproblems (selecting the best available service for an individual task) equal to the number of tasks in the workflow is constructed. The subproblems are constructed using users' QoS requirements. Normally, clients are unaware that whether their queries are implemented by single atomic service or by a composite service [4]. Hence, whatever QoS requirements they specify are meant for the entire workflow. The workflow level constraints are called as global constraints. While constructing subproblems, the global constraints are broken down into task-level constraints also known as local constraints with the condition that the aggregation of local constraints should always satisfy the global constraints. The QoS preferences need not be decomposed as they remain the same for workflow and tasks. In the following example, a user specifies his QoS requirements as *QoS constraints: cost <=\$5000 and latency<=*70 % and *QoS preferences: 80 % priority to latency and 20 % priority to cost.* In order to construct subproblems, it is essential to break down the global constraints into local constraints to meet the user's constraints, whereas user's QoS preferences will remain the same for both workflow level selection and task-level selection as clients are unaware of composition. Further, in some specific applications if at all the users are aware of composition and if they could specify constraints and preferences for individual tasks, then the given individual constraints and preferences will be taken into account during selection. Different utility functions will be constructed for different tasks, and best services for different tasks will be selected accordingly. After constructing subproblems, the independent subproblems are solved by finding the best available service for each subproblem. Ultimately, these selected best services are combined as the best available service combination for the given workflow.

1.1 Contributions

This section highlights the contributions of the work. There is a compelling need for selection of suitable services fairly quicker in order to meet dynamically changing business processes and service updates. In this work, a new and unique method based on *local selection* is contributed in the field of service composition for selecting suitable services as per QoS requirements. Though there are a few methods based on *local selection* available for service selection, a new method is proposed that decomposes the given global constraints in less computation time which is not based on MIP [2,3] but on extreme values of QoS attributes of service classes.

The advantages of this contribution include reduction in computation time, handling of combinational workflows with various execution patterns and 100 % guarantee for successful execution of all paths present in OR pattern. This contribution also assures improved efficiency in selection uniquely through heuristics-based constraint relaxation with user's trade-off. Further, the model can be integrated with any service-based application which involves QoS-based selection.

As a model, firstly, the contribution fills the gap of non-availability of efficient methods in terms of computation time and handling workflows with different execution patterns such as AND, OR and Loop. The model considers selecting a suitable service combination for the given workflow as the main problem and selecting suitable service for each task (present in the workflow) from its service class as a subproblem. It decomposes the given global constraints into local constraints in such a way that the aggregation of local constraints always meets the given global constraints. It solves the subproblems simultaneously and returns the combined solution as the best available service combination. For each service class, the model computes the local constraints based on extreme (minimum and maximum) values of QoS attributes of service class rather than the QoS values of individual services. Computing local constraints based on extreme values of QoS attributes of service class makes decomposition of constraints as independent of number of services in a service class, which is a unique feature. The computation time of decomposing constraints will not change with number of services which makes the model a better choice for applications having large service space.

Secondly, any local selection approach has an inherent limitation that under some situations, the method may fail to identify a suitable service combination based on local constraints in spite of availability of such combination. The proposed model contributes a heuristics-based approach to solve the problem by relaxing local constraints within the permitted limits.

Thirdly, a new aggregation function which does not take into account the probability factor associated with the paths of an OR execution pattern while computing aggregate value of response time is proposed. This new function assures 100 % guarantee for successful execution of each alternate path of an OR pattern when it is given a chance for execution in contrast to existing aggregation function [9,15,19,22], which considers probability factor of paths while computing aggregate value. With the existing function, it is not certain that every path of the OR pattern will succeed for execution as illustrated in Sect. 2.

From methodological perspective, the model performs service selection in two phases, viz., Constraint Decomposition Phase and Service Selection Phase. In Constraint Decomposition Phase, the given global constraints are decomposed into local constraints. This is done by first transforming the given workflow into sequential and then computing local constraints from the given global constraints, based on extreme values of QoS attributes of services classes. The model constructs subproblems by assigning local constraints to individual service classes. In Service Selection Phase, the model proposes an algorithm which finds suitable service for a task from its respective service class subject to its local constraints. The algorithm identifies best service from the available services (for a task) rather than optimal service. There is no guarantee that an optimal service will always exist. The model also describes the algorithmic steps involved in relaxing local constraints (still satisfying global constraints) to find a feasible solution.

From evaluation perspective as described in Sect. 4, with typical test collection, the proposed model is found to yield excellent time characteristics with sufficient accuracy (in terms of *utility_ratio*) when compared with existing approaches [2,15,21]. The model also describes how techniques such as heuristics-based constraint relaxation and user's trade-off among different QoS attributes help in increasing the efficiency of local selection.

The above features make the method more applicable to dynamic service composition applications in different domains such as e-health and e-tourism as per the example stated earlier in this section.

2 Related work

QoS-based local selection has been handled by many approaches. A local selection method presented in [20] mainly focuses on the decomposition of functional requirements. Local selection method presented in [12] uses LP technique to find optimal services under different modes, subjective weight mode, objective weight mode and subjective-objective weight mode. Its main focus is to handle user's preferences and objectivity of service quality characteristics. Another method presented in [4] finds the best candidate services for service composition by finding 'skyline services' for each class using dominance relationship. Though the work reduces the time complexity by carrying out the detection of skyline services as an offline job, its performance is affected by the number of constraints. An efficient method of decomposing constraints and distributed broker architecture is discussed in [2] for QoS-aware service selection. Though the method of decomposition of [2] is widely used by other research works [13,14,18,23,25], it has some inadequacy as discussed below.

In a service class, for each QoS attribute, there will be at least one service with minimum value and at least one service with maximum value. Hence, the value of a QoS attribute of service class ranges from a minimum to a maximum. While decomposing a global constraint, the method divides the range of QoS attribute into a set of quality levels. For each service class, the above method finds a particular quality level which has the highest benefit and assigns the same quality level as constraint for that service class. The benefit is computed using

$$p_{jk}^{z} = \frac{h(q_{jk}^{z})}{l} \cdot \frac{u(q_{jk}^{z})}{u_{\max}} \tag{1}$$

In (1), p_{jk}^{z} represents the benefit of using zth quality level of kth attribute of jth service class, $h(q_{jk}^{z})$ denotes the number of services that qualify the zth level, $u(q_{jk}^{z})$ denotes the highest utility value obtained by considering the qualified services of zth level, $u_{\max}$ denotes the highest utility obtained by considering all services in the jth service class and l denotes the number of services in jth service class. From (1), it is understood that the benefit of a quality level is influenced by number of services present in that level. When local constraints are fixed based on benefit, a quality level with less utility may be fixed as local constraint to a service class. In that case, the method fails to discover a service with high utility even though it exists (and hence satisfies a user less). So, the benefit of a quality level should not be influenced by the number of services that qualify that level. Further, the above approach handles only sequential workflow, and it is not applicable to the most widely used business execution patterns such as AND, OR and Loop patterns. Also, the method uses Mixed Integer Programming (MIP) to find local constraints which has poor time characteristics when number of services grows.

In [9,15,19,22], the OR execution pattern is discussed with respect to computation of QoS attributes. Let us consider an OR pattern, u containing k sequential paths, $P_1, P_2, \ldots, P_k$. Let the ith sequential path denoted by P_i contain m_i tasks, $t_1^i, t_2^i, \ldots, t_{m_i}^i$. Let p_i denote the probability of execution of P_i. Let $rt(t_j^i)$ denote the response time of jth task in P_i. Let $rt(P_i)$ denote the response time of P_i. The value of $rt(P_i)$ is calculated using

$$rt(Pi) = \sum_{j=1}^{mi} rt(t_j^i) \tag{2}$$

Let rt (u) denote the response time of u. The above approaches compute the '*expected response time*' of u using (3) and fix the computed value as $rt(u)$.

$$rt(u) = \sum_{i=1}^{k} pi \times rt(Pi) \tag{3}$$

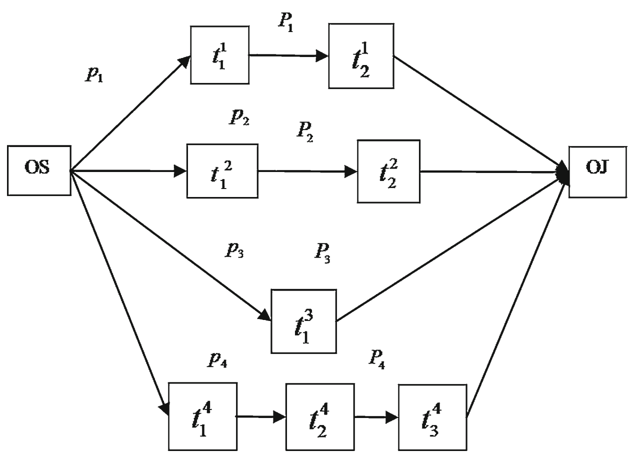

Fig. 2 An example OR unit

When response time of an OR pattern is computed based on expected value as given by (3), it is not certain that every path of the OR pattern will succeed for execution when a chance is given for execution. This limitation is illustrated using an example OR unit, u as given in Fig. 2.

In Fig. 2, 'OS' denotes OR Split and 'OJ' denotes OR Join. P_1, P_2, P_3 and P_4 denote the sequential paths present in u. P_1 consists of two tasks, t_1^1 and t_2^1. P_2 consists of two tasks, t_1^2 and t_2^2. P_3 consists of one task, t_1^3. P_4 consists of three tasks, t_1^4, t_2^4 and t_3^4. Based on run-time conditions, anyone of the sequential paths in u will be executed. Let p_1, p_2, p_3 and p_4 denote the probability of execution of P_1, P_2, P_3 and P_4, respectively. Let $rt(u)$ denote the response time of u. Now, consider typical values for probability of execution of different sequential paths and for response time (minimum response time) of different tasks as given in Tables 2 and 3, respectively.

According to (3), the value of $rt(u)$ is computed as 1,160. When this computed value is assigned as $rt(u)$, only P_1 and P_2 will succeed for execution. But P_3 and P_4 will fail because the response time of P_3 and P_4 (1,600 and 1,400, respectively) is greater than $rt(u)$. Such failures affect the reliability and predictability of workflows. The failure of paths P_3 and P_4 is due to the consideration of probability factors of the paths of the OR pattern during the computation of response time. So, the probability of a path in OR pattern is meant to represent the chance of choosing a path for execution. Probability factors should not be considered while computing QoS (response time).

Table 2 Probability of execution of different sequential paths in the example OR unit

ID of sequential path	Probability of execution
P_1	0.2
P_2	0.1
P_3	0.5
P_4	0.2

Table 3 Minimum response time of different tasks in the example OR unit

ID of task	Minimum response time
t_1^1	100
t_2^1	200
t_1^2	150
t_2^2	50
t_1^3	1,600
t_1^4	400
t_2^4	200
t_3^4	800

As a novel approach to find response time of an OR unit which alleviates the above failure, initially the response time of each sequential path is computed. Let $rt(P_i)$ denote the response time of ith sequential path. Let k be the number of paths in u. Let m_i denote the number of tasks present in ith sequential path. The value of $rt(P_i)$ is calculated using (2), and the value of $rt(u)$ is computed using

$$rt(u) = \max\{rt(Pi)\,|1 \le i \le k\} \tag{4}$$

If $rt(u)$ is computed according to (4), then all the paths present in the OR unit will have sufficient time for execution. Using (4), the value of $rt(u)$ for the example OR unit is computed as 1600. With this computation, the execution through any path of OR unit will become successful.

In any service-based application, it is very essential to allot the *minimum required time* for execution to all tasks/paths/ that are present in an OR pattern to ensure 100 % guarantee for successful execution of composite service.

There may be situations where the *minimum required time* itself may not satisfy the client's demand for response time. This means that a client may require a response time which is lesser than the *minimum required time*. Certainly, it is essential to consider QoS requirements of clients, but the *minimum time required* to meet 100 % successful execution cannot be comprised to meet user's QoS constraints. If we comprise the time allotment, the most probable path (and hence most probable workflow) itself will fail even with best case services. This is illustrated with the following example.

Consider that Mr. X wants to travel from a place, say, A to another place, say, B. Imagine that the travel can be performed using two modes of transport, namely flight and train. The probability of travel by the two modes of transport is 10 and 90 % respectively. Mr. X wants to book his ticket with a *Booking Service*. The example is realized with an OR unit as given in Fig. 3. Let u denote the OR pattern given in Fig. 3. The pattern consists of two sequential paths, P_1 and P_2. Each

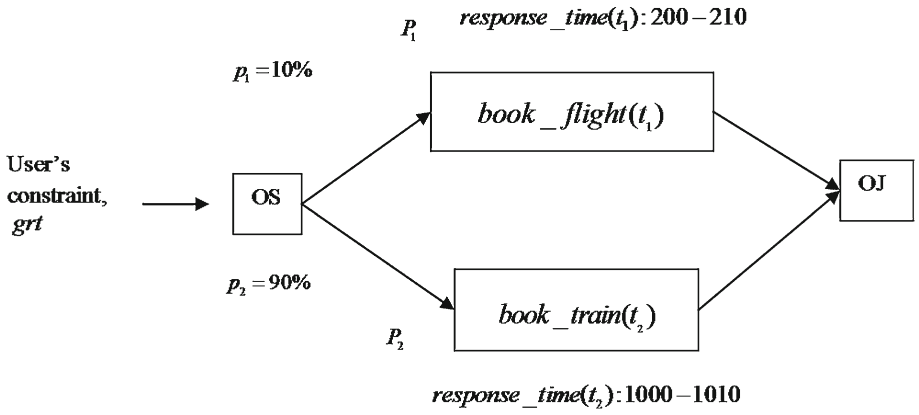

Fig. 3 Another OR pattern

path contains single task. The path P_1 contains the task t_1, and P_2 contains t_2. Corresponding to each task, multiple services are available to implement the functionality of that task, but with different QoS characteristics. Services which have similar functionality but different QoS corresponding to a task would form the service class of that task. Now, within a service class for each QoS attribute, there may be at least one service having minimum value for that attribute and there may be at least one service having maximum value of that attribute. Hence, the value of QoS attribute of a service class ranges from a minimum to maximum. Let t_i denote ith task. Let p_i, min_$rt(t_i)$ and max_$rt(t_i)$ denote the probability of execution, minimum response time and maximum response time of t_i. The values of probability of execution, minimum response time and maximum response time of different tasks are given in Table 4.

Let min_$rt(u)$ and max_$rt(u)$ denote the minimum response time and maximum response time of u. The values of min_$rt(u)$ and max_$rt(u)$ are computed using (3) (for simplicity, we call this method of computing response time as *conventional approach*), and proposed method of computing response time for an OR pattern using (4) is given in Table 5.

Table 4 Probability of execution, minimum response time and maximum response time of different tasks

Task	Probability	Minimum response time	Maximum response time
t_1	0.1	200	210
t_2	0.9	1,000	1,010

Table 5 Value of minimum response time and maximum response time of OR pattern

Extreme values of response time of OR pattern	Conventional method	Proposed method
min_$rt(u)$	920	1,000
max_$rt(u)$	930	1,010

When the values of min_$rt(u)$ and max_$rt(u)$ are fixed according to conventional approach (i.e., 920 and 930), the task t_2 which has 90 % probability for execution will fail for execution. This task cannot be executed even with best service offer with minimum response time (1,000).

Here, irrespective of whether the OR pattern can meet user's constraint of response time, the pattern fails for 90 % of its execution because of the insufficient allotment of time. Here, the reliability of the workflow will be lost. If we allot insufficient time to a workflow which may contain many OR patterns, we cannot even predict the success of the workflow which ultimately affects the client's satisfaction.

Hence, the first aspect to be considered is to allot the minimum required time which guarantees 100 % successful execution, and then, the second aspect is to be considered is how to fulfill the user's constraints.

Let grt denote the user's constraint of response time. Whether the OR execution pattern given in Fig. 3 is guaranteed for successful execution with respect to different cases of grt with the response time of the OR pattern computed using conventional and proposed methods are given in Table 6.

From Table 6, the proposed method of computing QoS values are found to satisfy the given user's constraints very well when compared to conventional method. Only when $grt < \text{min\_}rt(u)$, the execution is not guaranteed for 100 % and in this case, negotiation can be made with users for relaxing the constraint just to meet 100 % guarantee for successful execution with best service offers. For any other value of $grt \geq \text{min\_}rt(u)$, the OR pattern is found to be 100 % feasible with either best case or other services.

Another work [16] employs a hierarchical Petri nets-based approach to decompose global constraints into local constraints for different tasks in a general flow structure. But this method computes QoS attributes of an OR unit based on expected values and assigns constraints accordingly. Hence, all branches of an OR unit are not guaranteed to be successful.

Table 6 Percentage of guarantee for successful execution with respect to different cases of global constraint of response time

Different cases of *grt*	% of guarantee for successful execution	
	Conventional approach $min\_rt(u) = 920$; $max\_rt(u) = 930$	Proposed approach $min\_rt(u) = 1{,}000$; $max\_rt(u) = 1{,}010$
Case 1: $grt < min\_rt(u)$	90 % of execution fails	90 % of execution fails
Case 2: $grt = min\_rt(u)$	90 % of execution fails	100 % guarantee for successful execution.
Case 3: $min\_rt(u) > grt \leq max\_rt(u)$	90 % of execution fails	100 % guarantee for successful execution.

Based on the motivation from the literature survey, this work proposes an alternate service selection methodology through an efficient decomposition of QoS constraints and a new method of computing QoS values for an OR pattern.

3 Methodology

Global selection method is a time-consuming process, and it cannot be used in real-time situations. To reduce the time complexity of the global approach, the proposed methodology uses local selection method. The methodology has two phases, viz., Constraint Decomposition Phase and Service Selection Phase that are performed in sequence. In Constraint Decomposition Phase, the given QoS constraints are decomposed into task-level constraints in such a way that the aggregation of local constraints always meets the given global constraints. Using task-level constraints and the given QoS preferences subproblems are constructed. In Service Selection Phase, each subproblem is solved to select the best available service based on user-defined utility (constructed using the user's preferences over different QoS attributes) from its service class subject to its local constraints. Ultimately, the services selected for all the tasks are combined to produce the best available service combination for the given workflow.

3.1 Constraint decomposition phase

The structure of a workflow may vary from a sequential as in Fig. 4 to a combinational structure that contains various execution patterns such as AND, OR and Loop as in Fig. 5.

In Fig. 5, 'AS' represents AND Split, 'AJ' represents AND Join, 'OS' represents OR Split, 'OJ' represents OR Join, 'LS' represents Loop Start and 'LJ' denotes Loop Join. An execution pattern starts from AS to AJ forms an AND unit, an execution pattern starts from OS to OJ forms an OR unit and execution pattern starts from LS to LJ forms a Loop unit with number of iterations given in brackets. An AND/OR/Loop unit can contain sequential paths of tasks (such as t_4, t_5 and t_6), zero or more other AND, OR units and Loop structures (like OR unit of t_8 and t_9 in sequential combination with t_7, Loop of t_2 and t_3) in it.

The decomposition of global constraints into local constraints is dependent on the method of computing QoS attributes for various execution patterns present in a workflow. The execution patterns that exist in a workflow may vary from a simple sequential task to series of sequential tasks to AND/OR/Loop units. The AND/OR/Loop units may be simple or complex. Any simple AND or OR unit will contain only sequential paths. A simple Loop contains only one sequential path in it. A complex unit may contain other simple/complex units inside. The proposed method of decomposing con-

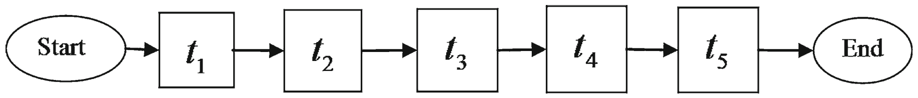

Fig. 4 Sequential workflow

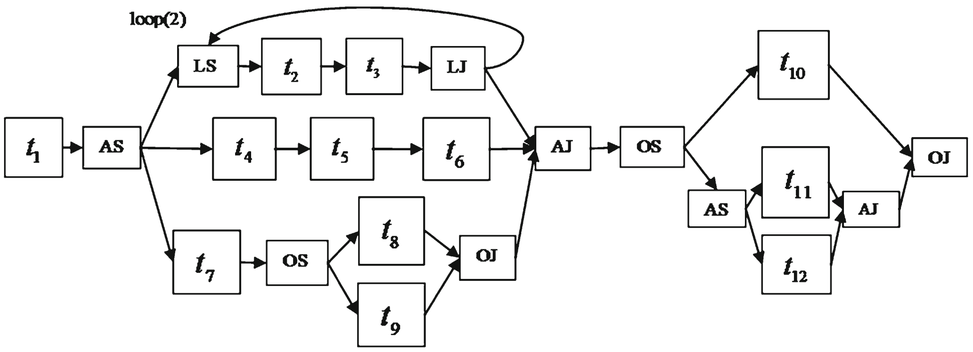

Fig. 5 Combinational workflow

Table 7 Computation of response time for various simple units

Type of simple unit	Computation of response time
Sequential path with 'n' tasks	$\sum_{j=1}^{n} rt(t_j)$
Simple AND unit with 'k' number of sequential paths	$\max\{rt(Pi)\|1 \le i \le k\}$
Simple OR unit with 'k'number of sequential paths	$\max\{rt(Pi)\|1 \le i \le k\}$
Simple Loop having one sequential path with 'z' tasks and 'm' iteration	$m \times \sum_{j=1}^{z} rt(tj)$

straints is described using the QoS attribute, 'response time'. Computation of response time for various simple units is given in Table 7. In Table 7, $rt(t_j)$ denotes response time of jth task, P_i denotes ith sequential path in AND/OR unit and $rt(P_i)$ denotes response time of P_i.

During decomposition of constraints, three tasks should be performed. As the workflow ranges from sequential to combinational workflow with complex execution patterns, the given workflow is converted into its equivalent sequential workflow firstly.

Secondly, before decomposition, it is necessary to check whether there exists at least one service combination that satisfies the given global constraint. This is accomplished using the best service from each service class. The best service from each service class is combined to form the best service combination. The QoS of the best service combination is computed and tested against the given global constraint. If the best service combination satisfies the given global constraint, then the given global constraint will be considered for decomposition. Otherwise, there will be no service combination available to implement the given workflow subject to the given constraints. In such case, the constraints will not be considered for decomposition, and the user can relax the constraints or he has to find services provided by yet other providers.

Thirdly, if a given constraint is found to be decomposable, it will be decomposed into local constraints and local constraints will be assigned to individual tasks. Further, to carry out the above tasks, the Constraint Decomposition Phase is split into three steps, namely conversion, decomposability check and decomposition of constraints.

3.1.1 Conversion

In this first step, the given workflow denoted by W is converted into sequential workflow denoted by W'. To illustrate the method of conversion, initially, the basic methods of conversion of simple AND, simple OR and simple Loop units are discussed. Any workflow can be converted into sequential workflow using these basic methods.

1) Conversion of Simple AND/OR/Loop Units (Basic methods)

Let us consider a simple unit u. The unit u may be a simple AND or a simple OR or a simple Loop. Let us consider u as a simple AND unit or a simple OR unit as in Fig. 6.

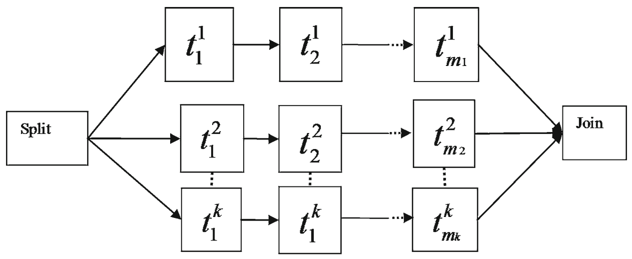

Fig. 6 Simple AND/OR unit

Let k be the number of sequential paths in u . Let ' P_i ' denote the ith sequential path in u. Let m_i, $1 \le i \le k$ be the number of tasks in P_i. Let t_j^i denote the jth task in P_i. The value of a QoS attribute of a task (or service class) varies from a minimum to a maximum as the service class contains many functionally similar services with varying QoS, and therefore, there will be a service with minimum value and a service with maximum value in the service class. Let min $\_rt(t_j^i)$ and max$_rt(t_j^i)$ denote the minimum response time and maximum response time of t_j^i. Let min$_rt(P_i)$ and max$_rt(P_i)$ denote the minimum response time and maximum response time of P_i.

For each $1 \le i \le k$, the values of min$_rt(P_i)$ and max$_rt(P_i)$ are calculated using (5) and (6).

$$\min\_rt(P_i) = \sum_{j=1}^{mi} \min\_rt(t_j^i) \tag{5}$$

$$\max\_rt(P_i) = \sum_{j=1}^{mi} \max\_rt(t_j^i) \tag{6}$$

Let min$_rt(u)$ and max$_rt(u)$ denote minimum response time and maximum response time of u. The values of min$_rt(u)$ and max$_rt(u)$ are calculated using (7) and (8).

$$\min\_rt(u) = max\{\min\_rt(P_i)\,|1 \le i \le k\} \tag{7}$$

$$\max\_rt(u) = max\{\max\_rt(P_i)\,|1 \le i \le k\} \tag{8}$$

Now, the given AND or OR unit is replaced by a sequential new-task whose minimum response time and maximum response time are equal to min$_rt(u)$ and max$_rt(u)$, respectively.

Consider u as a simple Loop that contains z sequential tasks as given in Fig. 7.

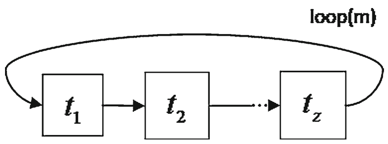

Fig. 7 Simple loop unit

Let 't_i' denote ith task in u. Let $\min\_rt(t_i)$ and $\max\_rt(t_i)$ denote the minimum response time and maximum response time of t_i. Let $\min\_rt(u)$ and $\max\_rt(u)$ denote the minimum response time and maximum response time of u. Let 'm' denote the number of iterations that u takes. The values of $\min\_rt(u)$ and $\max\_rt(u)$ are calculated using (9) and (10).

$$\min\_rt(u) = m \times \sum_{j=1}^{z} \min\_rt(t_j) \tag{9}$$

$$\max\_rt(u) = m \times \sum_{j=1}^{z} \max\_rt(t_j) \tag{10}$$

Now, u can be replaced by a sequential new-task whose minimum response time and maximum response time are equal to $\min\_rt(u)$ and $\max\_rt(u)$, respectively.

2) Conversion of given workflow into sequential workflow

Initially, all simple AND/OR/Loop units in the given workflow are converted into new-tasks using the basic methods. The resulting workflow may be a sequential or a combinational workflow. In the latter case, the conversion process is repeated until a sequential workflow is obtained. This sequential workflow is denoted by W', and this workflow contains two kinds of tasks old tasks (sequential tasks in the given workflow) and new-tasks (converted tasks).

3.1.2 Decomposability check

In decomposability check, the compliance of W' for its fulfillment against the given global constraint of response time is tested as follows.

Let $t_1, t_2, \ldots, t_m$ denote the old tasks in W'. Let $t'_1, t'_2, \ldots, t'_n$ be the new-tasks in W'. Let $\min\_rt(t_1), \min\_rt(t_2), \ldots, \min\_rt(t_m)$ denote minimum response time of 1st, 2nd,..., mth old tasks and $\min\_rt(t'_1), \min\_rt(t^1_2), \ldots, \min\_rt(t'_n)$ denote minimum response time of 1st, 2nd,..., nth new-tasks. Let $\max\_rt(t_1), \max\_rt(t_2), \ldots, \max\_rt(t_m)$ denote maximum response time of 1st, 2nd ,..., mth old tasks. Let $\max\_rt(t'_1), \max\_rt(t^1_2), \ldots, \max\_rt(t'_n)$ denote maximum response time of 1st, 2nd,..., nth sequential new-tasks. Let $\min\_rt(W')$ and $\max\_rt(W')$ denote the minimum response time and maximum response time of W' respectively. The values of $\min\_rt(W')$ and $\max\_rt(W')$ are found using (11) and (12).

$$\min\_rt(W') = \sum_{i=1}^{m} \min\_rt(t_i) + \sum_{i=1}^{n} \min\_rt(t'_i) \tag{11}$$

$$\max\_rt(W') = \sum_{i=1}^{m} \max\_rt(t_i) + \sum_{i=1}^{n} \max\_rt(t'_i) \tag{12}$$

Let 'grt' denote the given global constraint of response time. If $\min\_rt(W') > grt$, the given workflow cannot be implemented using the available services. In this case, the user can relax the constraints or he has to find services provided by yet other providers. If $\min\_rt(W') \leq grt$, W' is a feasible workflow. In the case of feasible workflow, the given global constraints are decomposed into task level constraints.

3.1.3 Decomposition of constraints

Based on the values of $\min\_rt(W')$ and $\max\_rt(W')$ and global constraint of response time, two local constraints of response time, namely lower bound constraint of response time and upper bound constraint of response time for each task in W' are computed. The assignment of lower and upper bound constraints to old tasks and new-tasks in W' is described below.

1) Assignment of constraints to old tasks

Let $\min\_rt(t_i)$ and $lcrt(t_i)$ denote the minimum response time and the lower bound constraint of response time of t_i. The value of $lcrt(t_i)$ is assigned using (13).

$$lcrt(t_i) = \min\_rt(t_i) \tag{13}$$

Let $ucrt(t_i)$ denote upper bound constraint of response time of t_i. The value of $ucrt(t_i)$ is computed for two different cases as follows.

Case 1: when $\min\_rt(W') = grt$

The value of $ucrt(t_i)$ is computed using (14).

$$ucrt(t_i) = \min\_rt(t_i) \tag{14}$$

Case 2: when $\min\_rt(W') < grt$

In this case, there are two subcases.

SubCase 2a: when $\max\_rt(W') \leq grt$

The value of $ucrt(t_i)$ is computed using (15)

$$ucrt(t_i) = \max\_rt(t_i) \tag{15}$$

SubCase 2b: when $\max\_rt(W') > grt$

The value of $ucrt(t_i)$ is computed using (16)

$$ucrt(t_i) = \frac{\max\_rt(t_i)}{\max\_rt(W')} \times grt \tag{16}$$

2) Assignment of constraints to new-tasks

In the case of a new-task, initially, the upper bound constraint of response time of a new-task is computed based on $\min\_rt(W')$, $\max\_rt(W')$ and grt. Let $ucrt(t'_i)$ denote the upper bound constraint of response time of t'_i. The value of $ucrt(t'_i)$ is computed for two different cases as follows.

Case 1: when $\min\_rt(W') = grt$
The value of $ucrt(t'_i)$ is computed using (17)

$$urct(t'_i) = \min\_rt(t'_i) \tag{17}$$

Case 2: when $\min\_rt(W') < grt$
In this case, there are two subcases.
SubCase 2a: when $\max\_rt(W') \leq grt$
The value of $ucrt(t'_i)$ is computed using

$$urct(t'_i) = \max\_rt(t'_i) \tag{18}$$

SubCase 2b: when $\max\_rt(W') > grt$
The value of $ucrt(t'_i)$ is computed using (19)

$$ucrt(t'_i) = \frac{\max\_rt(t'_i)}{\max\_rt(W')} \times grt \tag{19}$$

Now, corresponding to each new-task, there will be a simple unit. From the constraints of a new-task, the constraints of its corresponding simple unit are calculated. From the constraints of a simple unit, the constraints of tasks present in that unit will be computed.

To illustrate the assignment of constraints from a new-task to its respective unit and to the tasks present in the respective unit, a particular new-task t' is considered. Let $u(t')$ denote the simple unit corresponding to t'. This simple unit $u(t')$ may be a simple AND or a simple OR or a simple Loop. Let $ucrt(u(t'_i))$ denote the upper bound constraint of response time of $u(t')$. The value of $ucrt(u(t'_i))$ is computed using

$$ucrt(u(t')) = ucrt(t') \tag{20}$$

Consider $u(t')$ as a simple AND/OR unit. Let ‘k’ denote the number of sequential paths present in $u(t')$. Let $P_1, P2, \ldots, P_k$ denote first, second ,…, kth sequential path in $u(t')$. Let ‘m_i’ denote the number of tasks in an ith sequential path denoted by P_i. Let t^i_j denote the jth task in P_i. Let $\max\_rt(P_i)$ denote maximum response time of P_i. Let $lcrt(t^i_j)$ denote the lower bound constraint of response time of t^i_j. Let $ucrt(t^i_j)$ denote the upper bound constraint of response time of t^i_j. When a workflow is found to be feasible, then for each task, the minimum response time of that task is assigned as the lower bound constraint of the task. Thus, the value of $lcrt(t^i_j)$ is calculated using (21)

$$lcrt(t^i_j) = \min\_rt(t^i_j) \tag{21}$$

The value of $ucrt(t^i_j)$ is computed using (22)

$$ucrt(t^i_j) = \frac{\max\_rt(t^i_j)}{\max\_rt(P_i)} \times ucrt(u(t')) \tag{22}$$

Consider $u(t')$ as a simple Loop that contains ‘z’ tasks in the sequential path P. Let ‘m’ be the number of iterations that $u(t')$ takes. Let t_i denote ith task in P. Let $lcrt(t_i)$ denote the lower bound constraint of t_i. Let $ucrt(t_i)$ denote the upper bound constraint of t_i. The values of $lcrt(t_i)$ and $ucrt(t_i)$ are computed using (23) and (24).

$$lcrt(t_j) = \min\_rt(t_j) \tag{23}$$

$$ucrt(t_i) = \frac{\max\_rt(t_i)}{\max\_rt(P)} \times \frac{ucrt(u(t'))}{m} \tag{24}$$

Thus, the local constraints are assigned to all individual tasks in the given workflow.

3.2 Service selection phase

Service users have a set of QoS preferences based on their applications. The influence of a particular attribute on the user’s utility may be different from that of other attributes. Also, the attributes may be negative or positive. Negative attributes such as response time, latency and cost should be minimized. Positive attributes such as availability, scalability and reliability should be maximized. The preferences of user are modeled as a utility function using Simple Additive Weighting (SAW) technique [26]. The method presented in [2] is used for computing utility, and the utility of an ith service of jth service class is computed using

$$U(s_{ji}) = \sum_{k=1}^{r} \frac{Q_{\max}(j,k) - q_k(s_{ji})}{Q'_{\max}(k) - Q'_{\min}(k)} \times w_k \tag{25}$$

In (25), $Q'_{\max}(k)$ and $Q'_{\min}(k)$ denote the maximum and minimum values of kth QoS attribute of the given workflow. The value of $Q'_{\max}(k)$ is computed by aggregating the maximum values of kth attribute of all service classes by which the workflow is constructed. Similarly, the value of $Q'_{\min}(k)$ is computed by aggregating the minimum values of kth attribute of all service classes by which the workflow is constructed. $Q_{\max}(j,k)$ represents the maximum value of kth attribute of jth service class. $q_k(s_{ji})$ represents the value of kth attribute of ith service of jth service class, and w_k represents the weight of kth attribute. The utility function is subject to the condition, $\sum_{k=1}^{r} w_k = 1$ where r denote the number of attributes. The selection of the best available service for each task in the workflow is a subproblem, and it is formulated thus:

For each task t_j, select a service s_{ji} from its respective service class, S_j as the best available service subject to the conditions: $U(s_{ji})$ should be maximum of all $s_{ji} \in S_j$ and s_{ji} should satisfy the constraints of t_j.

In real-time service composition, the QoS constraints and preferences will be known only during querying time and precomputation of utility function, and its sorting is not possible for assisting quick selection of the best available service for a given task. Hence, an iterative searching procedure is developed to identify the best available service, and its pseudo code is given in Listing 1.

Listing 1 Pseudo code of searching procedure

```
Notations:
$S_j$ – $j^{th}$ Service class ; $C_j$ – Constraint of $j^{th}$ service class
$s_{ji}$ - $i^{th}$ service of $j^{th}$ service class ; $Q(s_{ji})$ – QoS attributes of $s_{ji}$
$U(s_{ji})$ – utility of $s_{ji}$ ; $s_{jb}$ –the best available service of $S_j$
$w_k$ - weight of $k^{th}$ attribute
Pseudocode: Best available service selection algorithm
Input: $S_j$, $C_j$, $Q(s_{ji})$ of all $s_{ji} \in S_j$ , $w_k$ for all attributes
Output: $s_{jb}$
Begin:
    $s_{jb} = null; maximum\_utility = 0;$
    For each service $s_{ji} \in S_j$
        If ( $Q(s_{ji})$ satisfies $C_j$ )
        {
            Compute $U(s_{ji})$
            If ( $U(s_{ji}) > maximum\_utility$ ) then { $maximum\_utility = U(s_{ji})$; $s_{jb} = s_{ji}$ }
        }
    End for
    Return $s_{jb}$
End
```

The best available services identified for all tasks of a given workflow are combined to produce the best available service combination.

Further, how the two phases of the proposed methodology work is illustrated with a simple example in Appendix A (supplementary material).

4 Experimentation

There are three objectives of experimentation. The first one is to find the computation time of the proposed method to select appropriate services for composition. The computation time is found out as the sum of computation time of Constraint Decomposition and Service Selection phases. The second one is to check the correctness of the proposed approach with standard approach. The third one is to compare the computation time of the proposed approach with other existing local approaches.

Toward experimentation, prototype as in Fig. 8 has been implemented and various experiments have been conducted.

The workflows from sequential to simple combinational are considered for testing the methodology. In this prototype, the input workflow is represented as a matrix denoted by M.

The steps involved in representing a simple combinational workflow as a matrix are given in Listing 2.

Let us consider a simple combinational workflow shown in Fig. 9. Let the workflow be denoted by W. According to the steps given in Listing 2, the details of tasks are captured in Table 8. There are 12 tasks in W. The tasks are numbered from 1 to 12. The workflow contains one AND unit, 2 OR units and one Loop. AND unit is numbered as 1.

Listing 2 Steps involved in representing a workflow as matrix

Step 1: Assign continuous numbering to tasks present in the workflow. The number assigned to a task becomes the ID of that task. Let an ith task be denoted by t_i where i denotes the ID of the task. For example, the tasks present in the workflow (please refer Fig. 9) are assigned IDs as in $t_1, t_2, t_3, \ldots t_{12}$. IDs are assigned from 1 to n where n denotes the number of tasks present in the workflow.
Step 2: The type of execution pattern in which a task is present is called its *unit type*. Let $ut(t_i)$ denote the unit type of ith task. For each t_i, $ut(t_i)$ can take values 1,2 and 3 according to AND, OR and Loop, respectively, in which t_i is present. If t_i is present as an individual task, $ut(t_i)$ is 4.
Step 3: Assign numbers to patterns. The number assigned to a pattern becomes the *unit ID* for the tasks present in that unit. In our numbering, we assign numbers to all patterns such that AND patterns receive continuous numbering starts from 1. Similarly, OR patterns and Loops receive continuous numbering starting from 1, i.e., AND units are numbered as 1,2,3 and so on and OR units are numbered as 1,2,3 and so on. Similarly, Loop patterns are numbered as 1,2,3…. When two units receive same number, they are differentiated by unit type. Now, for each task t_i, $u(t_i)$ denotes the unit ID of t_i. Further, for individual tasks, $u(t_i) = 0$
Step 4: For each pattern, assign continuous numbering starting from 1 to sequential paths present in that pattern. The number assigned to a sequential path becomes the *path ID* for the tasks present in that path. Let $sp(t_i)$ denote path ID of t_i. When two sequential paths receive same number, they are differentiated by unit ID and unit type
Step 5: Now, each task can be uniquely identified using unit type, unit ID and path ID.
Step 6: For each task t_i, if it is present in a loop, the number of iterations denoted by $it(t_i)$ takes an integer greater than 0. Otherwise, $it(i) = 0$. Let $\min\_rt(t_i)$, $\max\_rt(t_i)$, $lcrt(t_i)$ and $ucrt(t_i)$ denote the minimum response time, maximum response time, lower bound constraint of response time and upper bound constraint of response time of t_i, respectively. For each t_i, the values of $ut(t_i)$, $u(t_i)$, $sp(t_i)$ and $it(t_i)$ are captured from the given workflow. For each t_i, the values of $\min\_rt(t_i)$ and $\max\_rt(t_i)$ are also given as inputs.
Now, the given workflow is represented as a Matrix, M, such that $M = (M_{ij}, 1 \le i \le n, 1 \le j \le 9)$. The order of M is given as $n \times 9$ where 'n' denotes the total number of tasks present in the workflow. Let ith row of M denote ith task of the workflow. For each ith row, the values of task ID, $ut(t_i)$, $u(t_i)$, $sp(t_i)$, $\min\_rt(t_i)$, $\max\_rt(t_i)$, $it(t_i)$, $lcrt(t_i)$ and $ucrt(t_i)$ denote 1st, 2nd, 3rd, 4th, 5th, 6th, 7th, 8th and 9th columns of M. Given the above details for each t_i, the values of $lcrt(t_i)$ and $ucrt(t_i)$ are computed using the proposed method of decomposition and updated in the matrix.
(Please note: along with the above details of tasks as matrix, global constraints also will be given as inputs to decomposition phase as in Fig. 8. Also note that the values of columns 8 and 9 of the matrix are initialized to 0. At the end of Constraint Decomposition Phase, the values of lower and upper bound constraints of tasks are updated in 8th and 9th columns of the matrix respectively.)

The two OR units are numbered as 1 and 2. The Loop unit is numbered as 1. The AND unit contains 5 tasks, namely t_1, t_2, t_3, t_4 and t_5. Now, we have $ut(t_1) = 1; ut(t_2) = 1; ut(t_3) = 1; ut(t_4) = 1; and\, ut(t_5) = 1$. Also, we have $u(t_1) = 1; u(t_2) = 1; u(t_3) = 1; u(t_4) = 1; and u(t_5) = 1$.

There are two sequential paths in the AND unit 1, and they are numbered as 1 and 2. The tasks t_1, t_2 and t_3 are present in sequential path 1 and hence $sp(t_1) = 1; sp(t_2) = 1; sp(t_3) = 1$. The tasks t_4 and t_5 are present in sequential path 2 and hence $sp(t_4) = 2; sp(t_5) = 2$. The task t_6 is present in Loop pattern with $ut(t_6) = 3; u(t_6) = 1; sp(t_6) =$

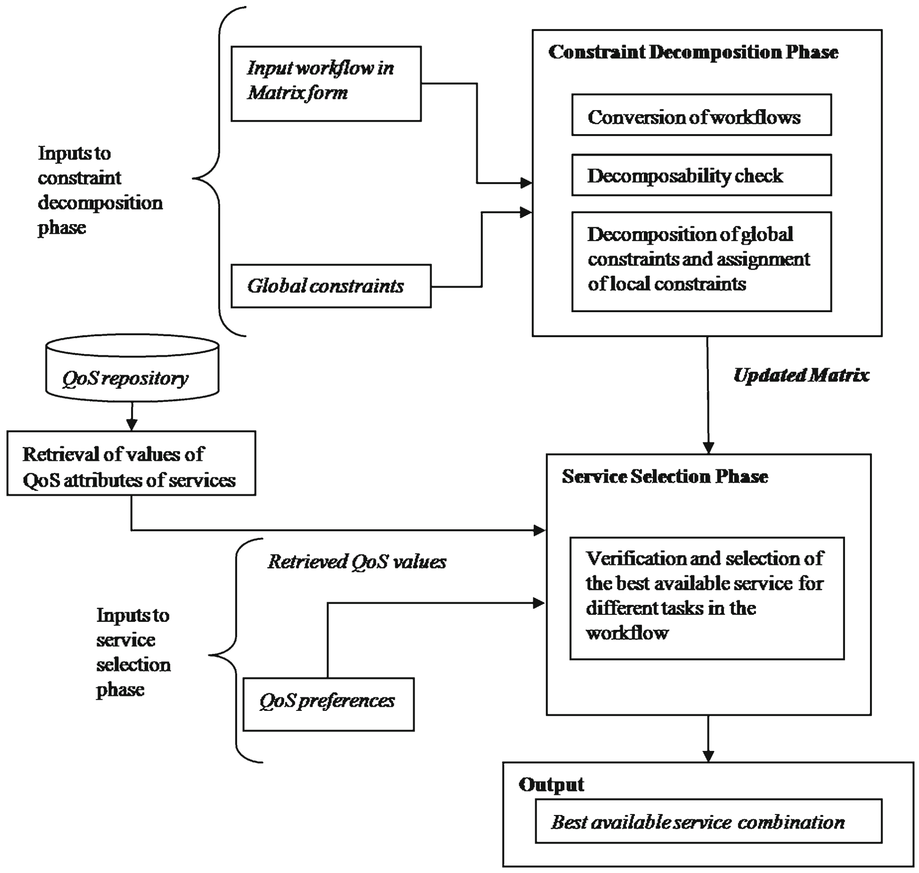
Inputs to constraint decomposition phase
Input workflow in Matrix form
Global constraints
Constraint Decomposition Phase
Conversion of workflows
Decomposability check
Decomposition of global constraints and assignment of local constraints
Updated Matrix
QoS repository
Retrieval of values of QoS attributes of services
Service Selection Phase
Retrieved QoS values
Inputs to service selection phase
QoS preferences
Verification and selection of the best available service for different tasks in the workflow
Output
Best available service combination

Fig. 8 Architecture of the prototype

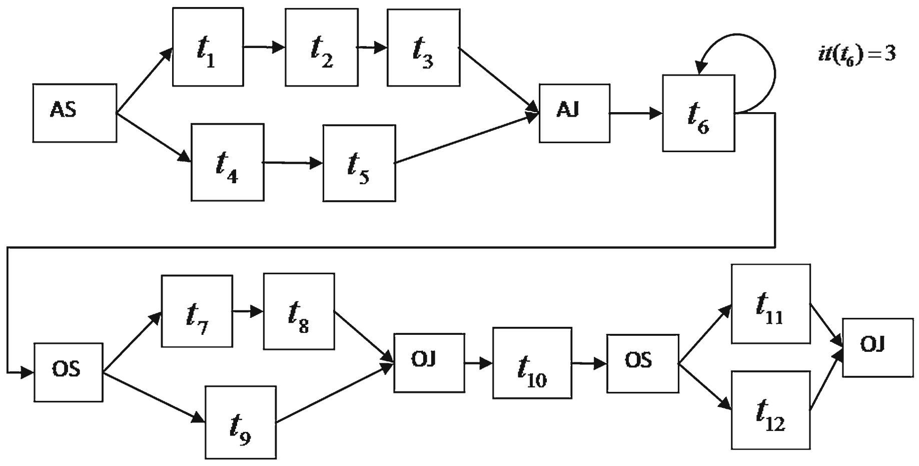

Fig. 9 An example workflow taken to illustrate matrix representation

1; $it(t_6) = 3$. The OR unit 1 contains 2 sequential paths numbered as 1 and 2. Using the steps given in Listing 2, we have unit types of t_7, t_8 and t_9 as $ut(t_7) = 2; ut(t_8) = 2; ut(t_9) = 2$. The unit IDs of t_7, t_8 and t_9 are given as $u(t_7) = 1; u(t_8) = 1; u(t_9) = 1$, and path IDs of t_7, t_8 and t_9 are given as $sp(t_7) = 1; sp(t_8) = 1; sp(t_9) = 2$. In a similar manner, details of other tasks are captured and given in Table 8. Also, note that assumed values are given for min_$rt(t_i)$ and max_$rt(t_i)$.

The details of tasks described in Table 8 can be mathematically visualized as a Matrix, M, such that $M = (M_{ij}, 1 \leq i \leq n, 1 \leq j \leq 9)$. That is the order of M is $n \times 9$ where 'n' denotes the total number of tasks present in W. Let the ith row of M denote the details of the ith task of W, and the columns of an ith row are admitted according to the columns of Table 8. The combinational workflow given in Fig. 9 is represented in its matrix form as in Fig. 10.

The workflow in Matrix form and the global constraints is given as inputs to Constraint Decomposition Phase. The given constraints are broken into local constraints and updated in the 8 and 9th columns of M. The updated matrix, user's QoS preferences and the values of QoS attributes

Table 8 Details of tasks present in the workflow

Task ID	$ut(t_i)$	$u(t_i)$	$sp(t_i)$	$\min\_rt(t_i)$	$\max\_rt(t_i)$	$it(t_i)$	$lcrt(t_i)$	$ucrt(t_i)$
1	1	1	1	10	100	0	0	0
2	1	1	1	20	120	0	0	0
3	1	1	1	5	150	0	0	0
4	1	1	2	50	200	0	0	0
5	1	1	2	10	300	0	0	0
6	3	1	1	14	125	3	0	0
7	2	1	1	20	200	0	0	0
8	2	1	1	14	150	0	0	0
9	2	1	2	26	200	0	0	0
10	4	0	0	12	100	0	0	0
11	2	2	1	14	130	0	0	0
12	2	2	2	30	140	0	0	0

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 10 & 100 & 0 & 0 & 0 \\
2 & 1 & 1 & 1 & 20 & 120 & 0 & 0 & 0 \\
3 & 1 & 1 & 1 & 5 & 150 & 0 & 0 & 0 \\
4 & 1 & 1 & 2 & 50 & 200 & 0 & 0 & 0 \\
5 & 1 & 1 & 2 & 10 & 300 & 0 & 0 & 0 \\
6 & 3 & 1 & 1 & 14 & 125 & 3 & 0 & 0 \\
7 & 2 & 1 & 1 & 20 & 200 & 0 & 0 & 0 \\
8 & 2 & 1 & 1 & 14 & 150 & 0 & 0 & 0 \\
9 & 2 & 1 & 2 & 26 & 200 & 0 & 0 & 0 \\
10 & 4 & 0 & 0 & 12 & 100 & 0 & 0 & 0 \\
11 & 2 & 2 & 1 & 14 & 130 & 0 & 0 & 0 \\
12 & 2 & 2 & 2 & 30 & 140 & 0 & 0 & 0
\end{pmatrix}$$

Fig. 10 Matrix representation of the example workflow

of individual services of different service classes (which is archived in QoS Repository) are given as input to Service Selection Phase. In Service Selection Phase, the best available services for different tasks of the workflow are identified simultaneously (using multiple threads) as per the search procedure given in Sect. 3.2.

Regarding test data for Constraint Decomposition Phase, in practice, the workflows are of different nature right from sequential workflow to combinational one. Hence, it is proposed to find the time taken for decomposing constraints to various cases such as sequential workflows, workflows having AND patterns, workflows having OR patterns, workflows have Loop patterns and workflows having combination of AND, OR and Loop patterns. This necessitates the construction of specific test collection for Constraint Decomposition Phase. So, a collection of specific workflows (having specified number of tasks, execution patterns, constraints, etc) according to requirements of experiments is constructed/synthesized and represented as matrices.

To study the computation time of Service Selection Phase, the QoS dataset from [Al-masri et al. http://www.uoguelph.ca/~qmahmoud/qws/index.html/] is used. The dataset contains 9 QoS attributes for 2500 real Web services. The QoS attributes include response time, availability, throughput, likelihood of success, reliability, compliance, best practices, latency and documentation. Using this dataset as base, QoS data have been created for a collection of 10,000 services.

Experiments are performed on a Laptop with Intel Pentium(R) Dual-Core, 2.20 GHz CPU, 3.0 GB memory and Windows 7 Ultimate Operating System. The pseudocode of the Constraint Decomposition Phase which is given in Appendix B (supplementary material) and the procedure for Service Selection Phase are implemented in Java 1.6 (J2SE) with Eclipse IDE.

4.1 Computation time of constraint decomposition phase

As the input workflow may vary from sequential to combinational one, time taken for decomposing constraints to various cases such as sequential workflow, simple AND/OR pattern, simple Loop and simple combinational workflow is discussed.

Case 1: The time taken for decomposing constraints to a sequential workflow by varying the number of service classes in the workflow from 10 to 100 is given in Fig. 11.

Case 2: The mechanism of decomposing and assigning constraints to a simple OR unit is same as that of a simple AND unit. Time taken for decomposing constraints to an AND/OR is calculated by varying the number of sequential paths in the pattern from 2 to 10. Time taken for decomposing constraints with respect to number of sequential paths is given in Fig. 12. In this example, each sequential path typically contains 2 service classes.

Case 3: Time taken for decomposing constraints to a simple Loop by varying number of service classes from 10 to 100 is given in Fig. 13.

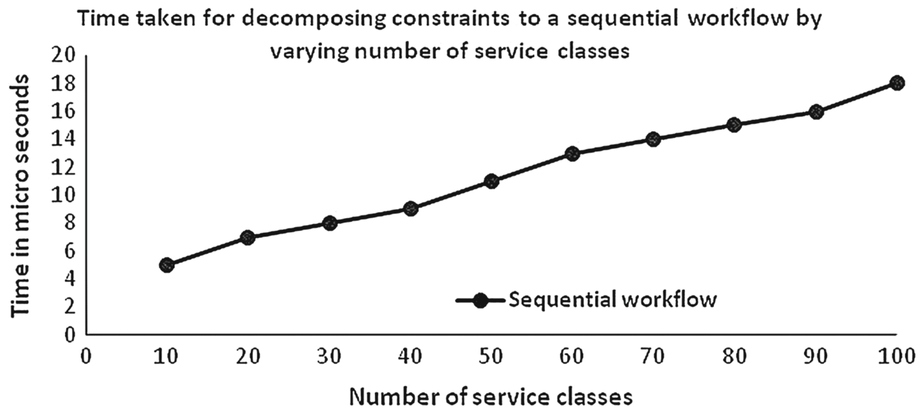

Fig. 11 Time taken for decomposing constraints to a sequential workflow by varying number of service classes

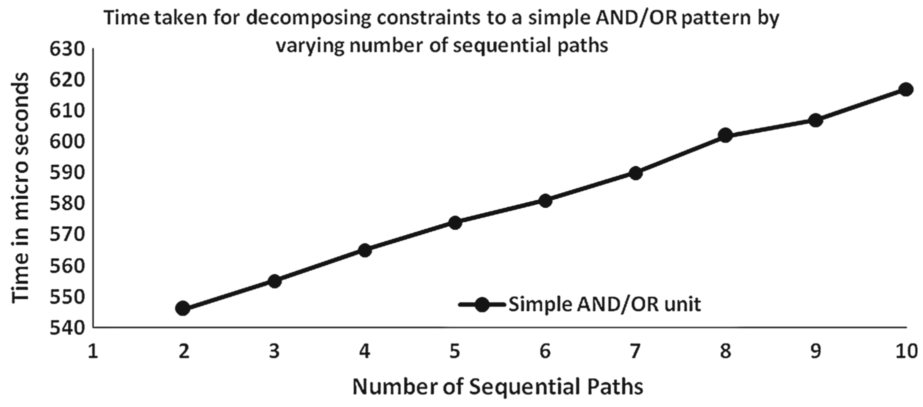

Fig. 12 Time taken for decomposing constraints to a simple AND/OR pattern by varying number of sequential paths

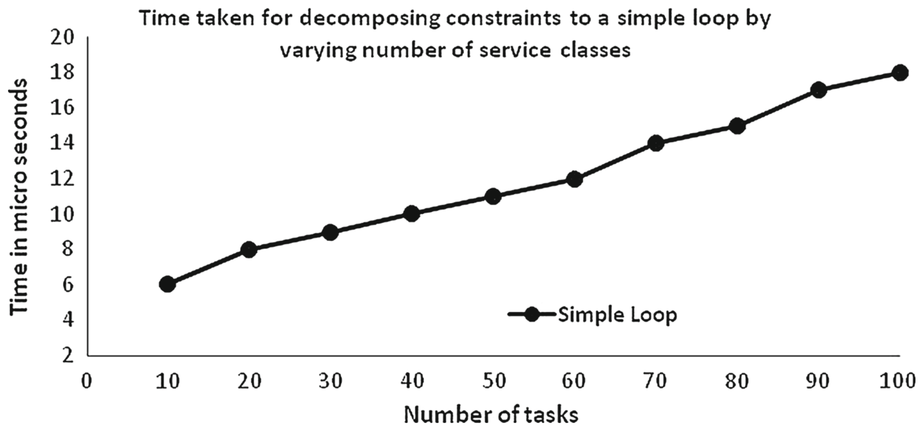

Fig. 13 Time taken for decomposing constraints to a simple loop by varying number of service classes

Case 4: The time taken for decomposing constraints to a simple combinational workflow which contains Simple AND, Simple OR and Simple Loop patterns is computed by varying the number of execution patterns is given in Fig. 14. The number of patterns in the workflow is varied from 3 to 30 in steps of 3. During experiment, the number of AND units is increased by 1, the number of OR units is increased by 1 and the number of Loops is increased by 1. In this case, each AND/OR units contains 2 sequential paths with each sequential path containing 10 service classes. Each simple Loop typically contains 10 service classes.

Case 5: Time taken for decomposing constraints to a simple combinational workflow by varying number of execution patterns for varied number of QoS constraints, denoted by ‘m’, is given in Fig. 15. For experimentation, negative attributes, namely response time, latency and cost, are chosen.

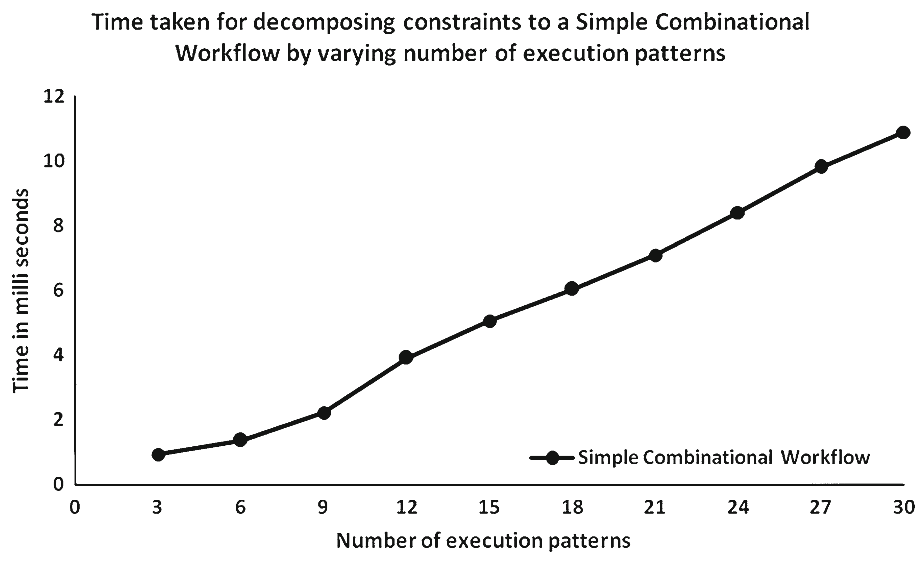

Fig. 14 Time taken for decomposing constraints to a simple combinational workflow by varying number of execution patterns

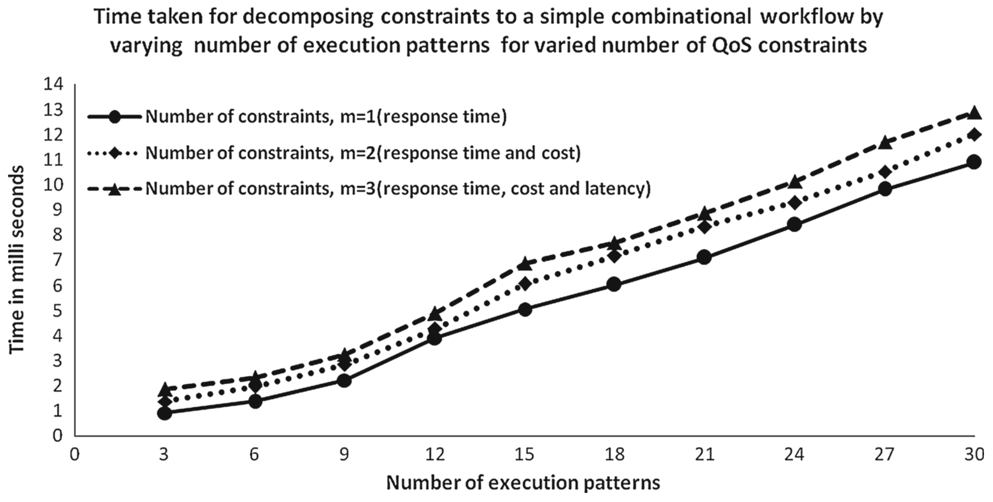

Fig. 15 Time taken for decomposing constraints to a simple combinational workflow by varying number of execution patterns for varied number of QoS constraints

The summary of time taken for decomposing constraints to different cases, namely sequential workflow, AND, OR, Loop and simple combinational workflow, is given in Table 9. In Table 9, N_{AND}, N_{OR}, N_{Loop} denotes the number of AND, OR and Loop patterns present in a workflow. N_{path} denotes the number of sequential paths present in an AND/OR/Loop. N_{sc} denotes the number of service classes present in a sequential workflow/AND/OR/Loop/combinational workflow. N_{cons} denotes the number of constraints and $t_{decompose}$ denotes the time taken for decomposing constraints.

From Table 9, it is found that the time required for decomposing constraints to an AND unit of 10 service classes (574 µs) is more than that of a sequential workflow having 10 service classes (5 µs). Because in case of AND unit, the minimum response time and maximum response time of the unit are computed, and before allocating constraints, the unit is converted into a single new-task. From the minimum response time and maximum response time of the new-task, the constraints of the corresponding sequential paths and tasks are computed. Due to the additional operations, the time involved in AND (or OR unit) is more than that of sequential workflow.

It is seen that time involved in fixing constraints to a Loop of ‘n’ service classes is same as that of a sequential workflow of same ‘n’ service classes because the conversion of simple Loop into sequential workflow requires only two extra operations. The first one is multiplication operation to multiply extreme values of response time of sequential path by the number of iterations of the Loop. The second one is reverse division operation while assigning constraints to individual tasks in the Loop (these operations consume time of only few nanoseconds). Further, it is observed that the increase in time taken for decomposing constraints to sequential workflow/AND/OR/Loop with respect to number of ser-

Table 9 Summary of computation time of constraint decomposition phase

N_{AND}	N_{OR}	N_{Loop}	N_{path}	N_{sc}	N_{cons}	$t_{decompose}$ of different cases (in ms)				
						Sequential	AND	OR	Loop	Combinational
–	–	–	1	10–100	1	0.005–0.018	–	–	–	–
1	–	–	2–10	4–20	1	–	0.546–0.617	–	–	–
–	1	–	2–10	4–20	1	–	–	0.546–0.617	–	–
–	–	1	1	10–100	1	–	–	–	0.005–0.018	–
1–10	1–10	1–10	2	50–500	1	–	–	–	–	0.924–10.885
1–10	1–10	1–10	2	50–500	2	–	–	–	–	1.385–12.018
1–10	1–10	1–10	2	50–500	3	–	–	–	–	1.865–12.895

vice classes/sequential paths is very gradual. From Fig. 15 and Table 9, the increase in decomposition time is found to vary by very small amount with respect to number of constraints. For example, for a workflow containing 500 tasks (10 AND patterns, 10 OR patterns and 10 Loops), the increase in decomposition time is only 2.01 ms (from 10.885 to 12.895) when the number of constraints is increased from 1 to 3.

4.2 Computation time of service selection phase

In Constraint Decomposition Phase, local constraints for different tasks are updated in the matrix M. QoS attributes of all services are available in the QoS repository as in Fig. 8. The priorities of user over different QoS attributes are captured. To find the utility of any ith service in jth service class, a utility function is constructed according to (25).

The updated matrix M, the utility function and the values of QoS attributes of all services of different service classes (from the QoS repository) are given as inputs to Service Selection Phase. Here, the QoS values of services should be retrieved from the QoS repository, and the values should be available to Service Selection Phase. The QoS values of services may be archived in different formats such as text file, excel file and database in the repository. The retrieval of QoS values from QoS repository to Service Selection Phase is an external process as in Fig. 8. Hence, the time taken for retrieval of QoS from repository to service selection phase is an external factor, and it should be excluded while calculating the computation time of Service Selection Phase.

In this work, the QoS values of test services are archived in a disk file. Though the retrieval of QoS is an external process, it is a prerequisite for service selection. Code has been developed to retrieve QoS attributes from file (disk) to an array (memory). Let $t_{retrieval}$ denote the time taken for retrieving QoS attributes from disk to memory. Let 'm' denote the number of constraints and l denote the number of services in a service class. To provide an insight into time requirement of retrieval process, the value of $t_{retrieval}$ with respect to l and m is given in Table 10.

Table 10 Value of $t_{retrieval}$ with respect to number of services and constraints

# of services	$t_{retrieval}$ (in ms)		
	m = 1	m = 2	m = 3
500	21.844	22.276	24.055
1,000	31.738	32.38	34.317
1,500	37.367	37.541	42.521
2,000	42.347	42.461	45.326
2,500	45.191	45.379	48.988
3,000	47.221	47.778	49.322
3,500	49.043	49.314	50.073
4,000	49.998	50.075	51.826
4,500	51.116	52.004	53.901
5,000	52.806	53.342	54.724
5,500	54.001	54.389	55.864
6,000	55.807	56.296	57.295
6,500	57.884	58.55	58.948
7,000	58.648	60.557	60.909
7,500	60.937	62.222	64.405
8,000	64.006	65.481	65.841
8,500	66.397	66.831	67.66
9,000	67.253	68.837	69.905
9,500	69.06	69.527	70.508
10,000	69.187	70.547	72.691

From Table 10, it is understood that $t_{retrieval}$ varies by a significant amount of time with respect to number of services. For example, when the number of services is increased from 500 to 10,000 (with m = 1), the retrieval time has increased from 21.844 to 69.187 ms. But for a given number of services, the variation in retrieval time with respect to number of constraints is less. For example, for 10,000 services, time taken for retrieving one attribute, two attributes and three attributes is 69.187, 70.547 and 72.691 ms, respectively.

From the above study, it is understood that a considerable amount of time of the order of few tens of milliseconds is involved in retrieving QoS. As the retrieval of QoS is a prerequisite for service selection, it is suggested that the retrieval of QoS values can be done prior to querying toward quick service selection. Also, to handle changes in QoS values of services, the values of QoS should be refreshed at specific intervals. In [22], the authors have monitored how frequently the QoS values of services change at different times of day with a collection of 39 service instances. The authors found that the QoS values of services tend to remain fixed from 13:30 to 21:30 h. Further, changes in QoS values have been observed at 1:33, 4:46, 7:39 and 12:29 h. The preloading of QoS attributes will reduce the time involved in selection of appropriate services. As in dynamic service composition, time is crucial; keeping literature [22] as evidence, we suggest refreshing the loaded QoS values once in 30 min.

During Service Selection, the QoS values of each service are verified for their fulfillment against the local constraints. This gives a set of services that satisfy the given constraints. From this set of services, the service which produces the maximum value for the user-defined utility function is selected as the best available service. Let t_{select} denote the time taken for verifying constraint fulfillment and selecting the best available service from a single service class. The service selection process for each task in independent, and hence, the process of service selection for different tasks is performed simultaneously using multiple threads.

The time taken for selecting the best available service from a service class with respect to l and m is found out, and the results are given in Table 11 and in Fig. 16. During experiment, the number of services is varied from 500 to 10,000, and m is varied from 1 to 3. QoS constraints, namely response time, cost and latency, are used for testing.

The summary of computation time of Service Selection Phase is given in Table 12.

In Table 12, $\text{N}_{\text{service}}$ denotes number of services in a service class, N_{cons} denotes number of constraints and t_{select} denotes the time required for verifying QoS values and finding the best available service. From Table 12, it is found that the increase in t_{select} with respect to number of constraints is very small and negligible. Also, from Table 11 and Fig. 16, it is seen that the increase in t_{select} with respect to number of services is linear.

4.3 Testing the correctness of the proposed approach

The correctness of the proposed method is evaluated by comparing the results obtained using the proposed method with the results obtained using global approach (taken as standard). We define *utility_ratio* as the ratio of utility obtained using the proposed approach to the utility obtained using the global approach. The value of *utility_ratio* exhibits how close the results of the proposed method are to the results of global approach. The value of *utility_ratio* is analyzed using sequential workflows by varying two parameters, namely number of service classes and number of services per service class. In global approach, initially one service from each service class is selected and a Composite Service (CS) is formed using the selected services. Then, the QoS of the CS is computed. The computed QoS is tested against the given global constraint. If a composite service fulfills the given global constraint, the utility of the composite service is computed. The utility of a composite service, denoted by U_{CS}, is computed as in [2], using (26)

$$U_{CS} = \sum_{k=1}^{r} \frac{Q'_{\max}(k) - q_k(CS)}{Q'_{\max}(k) - Q'_{\min}(k)} \times w_k \tag{26}$$

In (26), U_{CS} represents the utility obtained using global approach, $Q'_{\max}(k)$ represents sum of maximum values of k th attribute of all service classes involved in implementing a workflow and $Q'_{\min}(k)$ represents the sum of minimum values of kth attribute of all service classes involved in implementing a workflow, $q_k(CS)$ denotes value of k th attribute of CS and w_k represents the weight of k th attribute. In the proposed approach, the utility of an ith service of jth service class is computed using (25). The service which gives the highest utility is selected as the best available service for

Table 11 Time taken for selecting the best available service from a service class with respect to number of services for varied number of QoS constraints

# of services	t_{select} (in ms)		
	m = 1	m = 2	m = 3
500	0.019	0.020	0.020
1,000	0.034	0.034	0.034
1,500	0.051	0.051	0.051
2,000	0.066	0.066	0.066
2,500	0.083	0.083	0.086
3,000	0.098	0.099	0.099
3,500	0.115	0.115	0.116
4,000	0.129	0.131	0.131
4,500	0.146	0.146	0.146
5,000	0.163	0.163	0.163
5,500	0.176	0.179	0.179
6,000	0.192	0.194	0.198
6,500	0.206	0.209	0.210
7,000	0.238	0.238	0.243
7,500	0.251	0.254	0.252
8,000	0.264	0.268	0.271
8,500	0.277	0.282	0.291
9,000	0.295	0.305	0.308
9,500	0.319	0.319	0.321
10,000	0.327	0.330	0.331

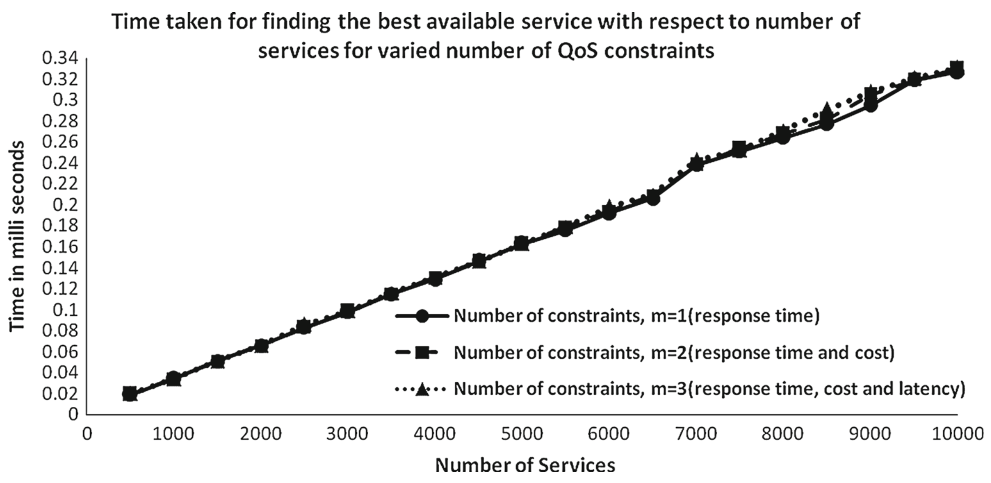

Fig. 16 Time taken for selecting the best available service from a service class with respect to number of services for varied number of QoS constraints

Table 12 Summary of computation time of service selection phase

N_{service}	N_{cons}	t_{select} in μs
500–10,000	1	19–327
500–10,000	2	20–330
500–10,000	3	20–331

a service class. Let s_{jb} denote the best available service of jth service class. Let $U(s_{jb})$ denote the utility of the best available service of jth service class. The utility of the best available service combination of a workflow with ‘n’ service classes, denoted by $U_{proposed}$, is computed using

$$U_{proposed} = \sum_{j=1}^{n} U(s_{jb}) \tag{27}$$

The value of *utility_ratio* is computed using

$$utility\_ratio = (U_{proposed}/U_{CS}) \times 100 \tag{28}$$

The value of *utility_ratio* computed by varying the number of service classes (with number of services per service class is fixed as 10) is given in Table 13.

From Table 13, the average value of *utility_ratio* by varying the number of service classes is taken as 99.5 %. Utility obtained using proposed and local approaches by varying number of service classes from 2 to 12 is given in Fig. 17.

Further, the value of *utility_ratio* by varying the number of services in a service class from 200 to 2,000 in steps of 200 is given in Table 14. Utility obtained using proposed and local approaches by varying the number of services from 200 to 2,000 (with number of service classes fixed as 5) is given in Fig. 18. The average *utility_ratio* by varying the number of services is found to be 99.86 %.

Table 13 Value of *utility_ratio* by varying number of service classes

# of service classes	U_{CS}	$U_{proposed}$	*utility_ratio* (in %)
2	0.96782	0.96782	100
3	0.958363	0.958363	100
4	0.990642	0.990642	100
5	0.970998	0.970998	100
6	0.98234	0.973491	99.09919
7	0.956581	0.947543	99.0552
8	0.940996	0.940996	100
9	0.965482	0.954574	98.87023
10	0.96348	0.955561	99.17802
11	0.984236	0.984236	100
12	0.971234	0.95543	98.37279

From the comparison results given in Tables 13 and 14, the accuracy of the proposed approach is found to be good.

4.3.1 *Computation time of proposed approach versus global approach*

As global approach is taken as the standard to validate the correctness of the proposed approach, the computational performance of proposed approach is also compared with global approach with respect to number of service classes and number of services for varied number of QoS constraints/attributes.

Firstly, by fixing the number of services classes (denoted by ‘n’) and number of services per service class (denoted by ‘l’), the computation time of global and proposed approaches for varied number of QoS constraints is found out as in Table 15. During this test, ‘n’ is kept as 5 and l is kept as 100. The number of constraints is varied from 1 to 3. QoS

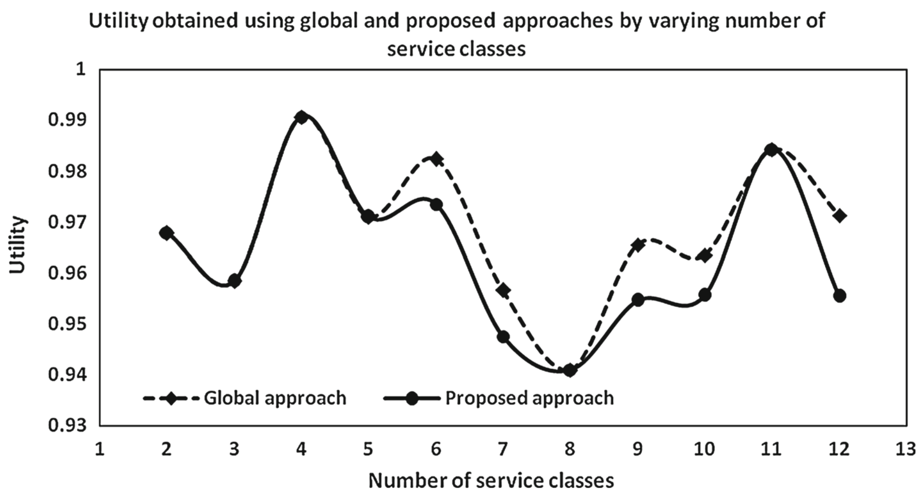

Fig. 17 Utility obtained using global and proposed approaches by varying number of service classes

Table 14 Utility obtained using global and proposed approaches by varying number of services

Number of services	Utility		*utility_ratio* (in %)
	Global approach	Proposed approach	
200	0.989876	0.984587	99.46569
400	0.992145	0.992145	100
600	0.991698	0.986792	99.50529
800	0.989654	0.989654	100
1,000	0.990991	0.990991	100
1,200	0.986789	0.986789	100
1,400	0.990124	0.987865	99.77185
1,600	0.993451	0.992188	99.87287
1,800	0.980976	0.980976	100
2,000	0.994304	0.994304	100

constraints, namely response time, cost and latency, are used. The graphs showing the computation time of global and proposed approaches with respect to number of constraints are given in Figs. 19 and 20, respectively.

From Figs. 19 and 20, in both approaches, the computation time increases linearly with respect to number of QoS constraints. Further, from Table 15, the computation time of proposed approach with respect to number of QoS constraints (for a fixed number of service classes and fixed number of services) is found to vary very slowly (from 10 to 14μs) when compared to that of global approach (from 221 to 360 s).

Secondly, the computation time of global and proposed approaches with respect to number of services for varied number of QoS constraints is given in Table 16. During testing, the number of service classes is fixed as 5. The number of services per service class is varied from 40 to 110. QoS constraints, namely response time, cost and latency, are used. The graphs showing the computation time of global and proposed approaches with respect to number of services for varied number of QoS constraints are given in Figs. 21 and 22, respectively.

From Table 16 and Fig. 21, the computation time of global approach with respect to number of services for varied number of QoS constraints is found to increase exponentially, and the cause of exponential time characteristics arises from the

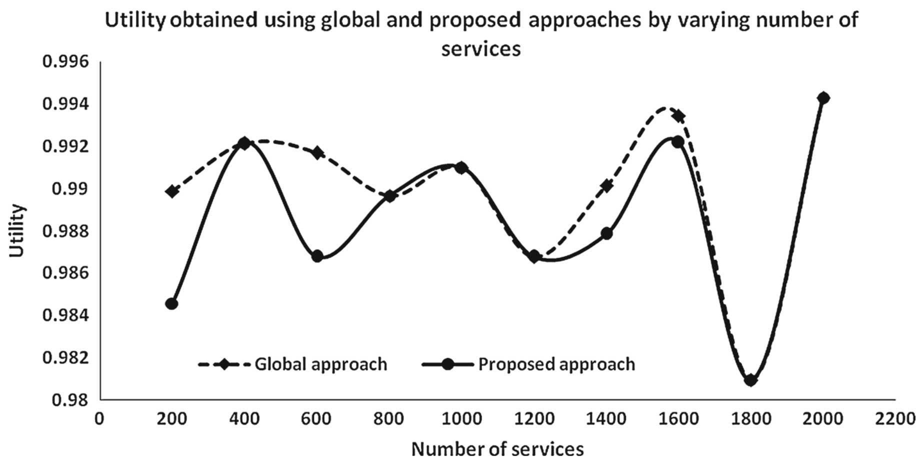

Fig. 18 Utility obtained using global and proposed approaches by varying number of services

Table 15 Computation time of proposed and global approaches for varied number of QoS constraints (for fixed n and l)

Number of constraints (m)	Global approach	Proposed approach		
	Computation time in seconds	Time in μs		
		Time of decomposition	Time of selection	Computation time
1	221	6	4	10
2	288	7	5	12
3	360	8	6	14

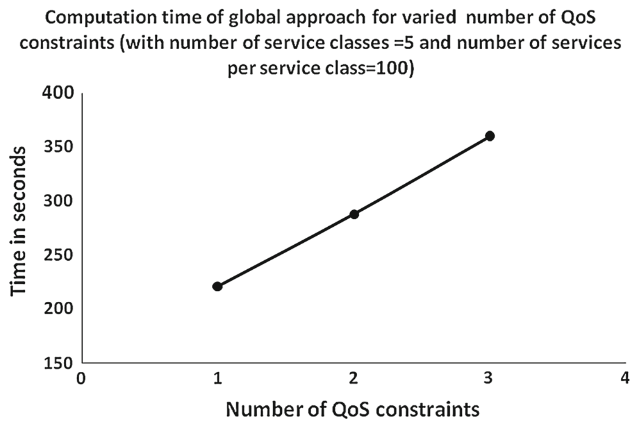

Fig. 19 Time taken by global approach for varied number of QoS constraints (by fixing n and l)

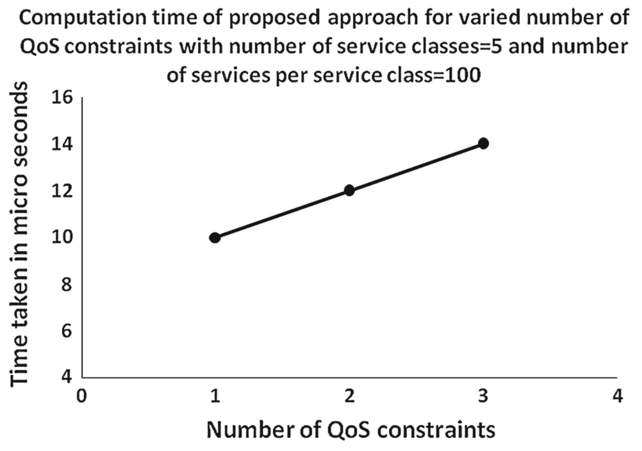

Fig. 20 Time taken by proposed approach for varied number of QoS constraints (by fixing n and l)

number of combinations involved in selection (and not from number of QoS constraints as seen from Fig. 19), whereas from Table 16 and Fig. 22, the computation time of proposed approach with respect to number of services for varied number of QoS constraints is found to increase very slowly.

Thirdly, the computational performance of proposed approach is compared with global approach with respect to number of service classes for varied number of QoS constraints as in Table 17. The number of services is fixed as 20. Further, the graphs showing the computation time of global and proposed approaches with respect to number of service classes are given in Figs. 23 and 24, respectively.

From Table 17 and Fig. 23, the computation time of global approach is found to increase exponentially with number of service classes. The exponential increase is due to exponential increase in number of combinations to be searched for selection, whereas from Table 17 and Fig. 24, the computation time of proposed approach is found to increase very slowly with respect to number of service classes.

4.4 Comparison of the proposed approach with existing local approaches

To compare the computation time of the proposed approach with other existing approaches, a comparative study is undertaken as follows.

- Comparison with Alrifai et al. Approach—there are a set of approaches [2,13,14,18,23,25] with same method of decomposition for QoS-based local service selection. The method of decomposing constraints of [13,14,18, 23,25] is based on [2]. Hence, at first, Alrifai et al. [2] has been chosen to compare with the proposed approach.
- Comparison with Lianyong Qi et al. Approach—the method proposed by Lianyong Qi et al. in [21] is similar to [2], but it reduces the candidate search space of composition with a heuristic solution called *Local Optimization and Enumeration Method*, which filters the numerous candidates, say 'l' candidates corresponding to each task into 'h' promising candidates. So, the method [21] is chosen for comparison with the proposed approach.
- Comparison with Freddy Lecue et al. Approach—the approach proposed in [15] addresses the scalability issue of service composition by selecting composition that satisfies a set of constraints rather than the one which produces optimal utility. So, we propose to compare the proposed approach with [15] also.

Table 16 Computation time of proposed and global approaches with respect to number of services for varied number of QoS constraints

Number of services per service class	Global approach			Proposed approach		
	Computation time in seconds			Computation time in μs		
	m = 1	m = 2	m = 3	m = 1	m = 2	m = 3
40	2	2	3	20	24	29
50	6	8	11	24	28	34
60	15	22	28	27	34	40
70	33	47	61	31	39	45
80	65	93	120	35	43	53
90	118	167	216	38	48	59
100	200	284	360	42	53	64
110	339	470	604	49	59	68

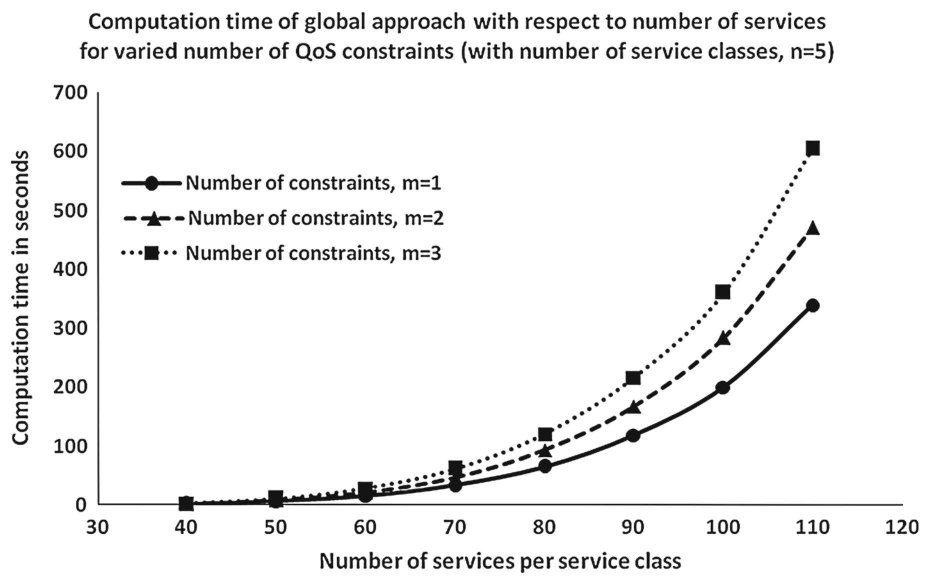

Fig. 21 Time taken by global approach with respect to number of services for varied number of QoS constraints

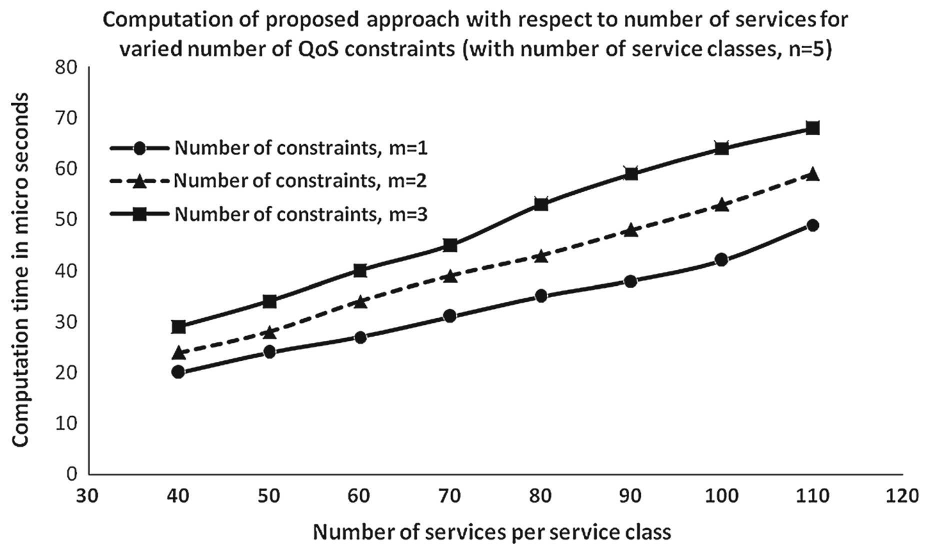

Fig. 22 Time taken by proposed approach for different m by varying number of services

Table 17 Computation time of global and proposed approaches with respect to number of service classes for varied number of QoS constraints

Number of service classes	Global approach			Proposed approach		
	Computation time in seconds			Computation time in μs		
	m = 1	m = 2	m = 3	m = 1	m = 2	m = 3
2	0.000033	0.000054	0.000064	12	13	14
3	0.000558	0.000932	0.001212	14	17	18
4	0.011792	0.015201	0.17364	16	19	21
5	0.07295	1.0337	1.33741	18	20	22
6	1	2	3	20	21	23
7	33	54	67	21	23	25
8	805	1158	1497	22	24	26

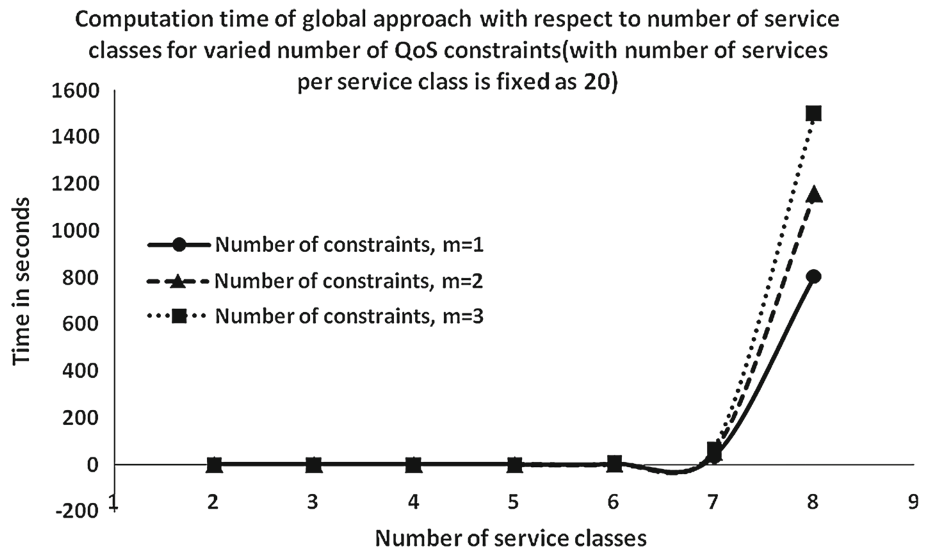

Fig. 23 Time taken by global approach by varying number of service classes for varied number of QoS constraints

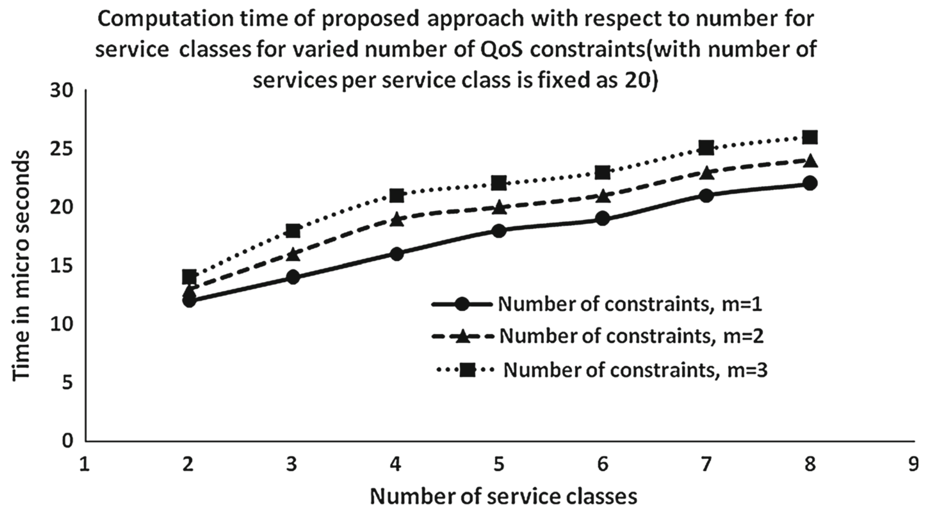

Fig. 24 Time taken by proposed approach by varying number of service classes for varied number of QoS constraints

- Comparison of time taken for computing QoS for an OR pattern— further, the proposed approach uses a new method of computing QoS for OR pattern in contrast to the approaches such as [9,15,19,22], which use the conventional method of computing expected values of QoS attributes of an OR pattern. But these approaches do not focus on time taken for computing QoS values, an experiment is conducted to compare the time taken for computing QoS values of an OR pattern using conventional and the proposed methods.

Further, the detailed experimentation is discussed below.

4.4.1 *Proposed approach versus Alrifai et al. approach*

Two experiments have been conducted for comparison. The experimental conditions similar to Alrifai et al. approach [2] have been used for the comparative study. Let ‘n’ denote the number of service classes and ‘m’ denote the number of constraints. In the first experiment, the values of ‘n’ and ‘m’ are fixed as 10 and 3, respectively, and the number of services in each service class is varied from 100 to 2,000 in steps of 100. The computation time of the proposed approach, denoted by $t_{proposed}$, is computed using (29).

$$t_{proposed} = t_{decompose} + t_{select} \tag{29}$$

The computation time of the proposed approach with respect to number of services in a service class is given in Table 18. From Table 18, it is found that the time taken for decomposing constraints ($t_{decompose}$) is only 0.009 ms and it is independent of number of services in a service class. The time involved in identifying the best available service(t_{select}) is ranging from 0.005 to 0.066 ms when the number of services is varied from 100 to 2,000. The computation time of the proposed approach with respect to number of services is compared to

Table 18 Computation time of the proposed approach with respect to number of services

# of services	Computation time in ms		
	$t_{decompose}$	t_{select}	$t_{proposed}$
100	0.009	0.005	0.014
200	0.009	0.008	0.017
300	0.009	0.012	0.021
400	0.009	0.015	0.024
500	0.009	0.020	0.029
600	0.009	0.023	0.032
700	0.009	0.024	0.033
800	0.009	0.028	0.037
900	0.009	0.031	0.040
1,000	0.009	0.034	0.043
1,100	0.009	0.037	0.046
1,200	0.009	0.040	0.049
1,300	0.009	0.043	0.052
1,400	0.009	0.048	0.057
1,500	0.009	0.051	0.060
1,600	0.009	0.054	0.063
1,700	0.009	0.058	0.067
1,800	0.009	0.060	0.069
1,900	0.009	0.063	0.072
2,000	0.009	0.066	0.075

values interpreted from Alrifai et al. [2], and the comparison results of Experiment 1 are given in Table 19.

From Table 19, the increase in computation time of the proposed approach with number of services is found to be very less and negligible when compared to that of Alrifai et al. [2]. Here, the drastic reduction in computation of local constraints arises from the method of computing local constraints which is based on extreme values of QoS attributes of service classes rather than the values of QoS attributes of individual services.

In the second experiment, the number of services per service class, denoted by ‘*l*’, is fixed as 500. The value of ‘m’ is fixed as 3. The value of ‘n’ is varied from 10 to 100 in steps of 10. The computation time of the proposed approach with respect to number of service classes is shown in Table 20.

The computation time of the proposed approach with respect to number of service classes is compared with the values interpreted from Alrifai et al. [2]. The comparison results of Experiment 2 are given in Table 21.

From Table 21, it is understood that the time variation in Alrifai et al. [2] ranges from approximately 500 ms to approximately 20,000 ms, whereas time variation in the proposed approach ranges from 0.029 to 0.070 ms. The time variation in the proposed approach is very small and negligible when compared to that of Alrifai et al. [2]. Here also, the drastic reduction in computation of local constraints is due to the method of computing constraints, which is based on extreme values of QoS classes rather than QoS values of individual services.

4.4.2 *Proposed approach versus Lianyong Qi et al. approach*

The method proposed in [21] is similar to [2], but it reduces the candidate search space of composition with a heuristic solution called *Local Optimization and Enumeration Method*, which filters the numerous candidates, say, ‘*l*’ candidates corresponding to each task into ‘*h*’ promising candidates. For an ith task, the range of say jth attribute is divided into‘d’ quality levels, namely $q_{ij}^1, q_{ij}^2, \ldots q_{ij}^d$ similar to [2]. Each level $q_{ij}^z | 1 \leq z \leq d$ represents a local constraint. For negative attributes, smaller q_{ij}^z produces more satisfactory utility to users, and for positive attributes, larger q_{ij}^z produces more satisfactory utility to users. Hence, in this method, initially for negative attributes, the constraints are fixed as $[0 - q_{ij}^z]$, and for positive attributes, the constraints are fixed as $[q_{ij}^z - \infty]$, and promising candidates are chosen with these constraints. Then, the constraints are reassigned from their previous levels to next level (i.e., for negative attributes, $[0 - q_{ij}^{z+1}]$, and for positive attributes, $[q_{ij}^{z-1} - \infty]$) to find out further promising candidates. The process is repeated to obtain ‘*h*’ *promising candidates*. Then,

Table 19 Comparison results of Experiment 1

# of service classes	# of constraints	# of services	Computation time in ms	
			Alrifai et al. approach	Proposed approach
10	3	100–2,000	50–600	0.014–0.075

Table 20 Computation time of the proposed approach with respect to number of service classes (with experimental conditions similar to Alrifai et al. approach)

# of service classes (n)	Computation time in ms		
	$t_{decompose}$	t_{select}	$t_{proposed}$
10	0.009	0.020	0.029
20	0.014	0.020	0.034
30	0.018	0.020	0.038
40	0.022	0.020	0.042
50	0.027	0.020	0.047
60	0.032	0.020	0.052
70	0.036	0.020	0.056
80	0.041	0.020	0.061
90	0.046	0.020	0.066
100	0.050	0.020	0.070

Table 22 Computation time of the proposed approach with respect to number of service classes (with experimental conditions similar to Lianyong Qi et al. approach)

# of service classes ('n')	Computation time in ms		
	$t_{decompose}$	t_{select}	$t_{proposed}$
5	0.007	0.003	0.010
10	0.011	0.003	0.014
15	0.013	0.003	0.016
20	0.017	0.003	0.020
25	0.020	0.003	0.023

with 'h'candidates in each task, near-to-optimal selection is made. In this manner, in [21] the search space is reduced from 'l' candidates to 'h' promising candidates. Further, the approach presented in [21] handles sequential workflows.

To compare the computation time of proposed approach with that of [21], experimental conditions similar to Lianyong Qi et al. [21] have been used. Let 'n' denote the number of service classes and 'm' denote the number of constraints. The value of 'n' is varied from 5 to 25. The value of 'm' is fixed as 4. The value of 'l' is fixed as 50. The computation time of the proposed approach $t_{proposed}$ is computed using (29) and given in Table 22.

The computation time of the proposed approach with respect to number of service classes is compared with values interpreted from Lianyong Qi et al et al. [21], and the comparison results are given in Table 23.

Another experiment is conducted to compare the computation time of the proposed approach with Lianyong Qi et al. approach by varying the number of services from 50 to 500 and keeping the number of service classes as 5 and number of constraints as 4. Computation time of the proposed approach obtained by varying number of services with experimental conditions similar to Lianyong Qi et al. approach [21] is given in Table 24.

The computation time of the proposed approach with respect to number of services is compared with values interpreted from Lianyong Qi et al. [21], and the comparison results are given in Table 25.

When we compare the computation time of Alrifai et al. method [2] with Lianyong Qi et al method [21], the computation time of Lianyong Qi et al. method [21] with respect to number of services is found to be better than Alrifai et al. method [2]. The reason must be Lianyong Qi et al. method [21] selects appropriate service combination from 'h' promising candidates rather than from 'l' candidates as in Alrifai et al. method [2]. Still, the computation time of Lianyong Qi et al. approach [21] with respect to number of service classes is quite high of the order of 10^3 ms, whereas the proposed method outperforms both Alrifai et al. method [2] and Lianyong Qi et al. method [21] in terms of computation time in our experimentation with given set of data (please refer Tables 19, 21, 23 and 25). The primary reason for low computation time is that the decomposition phase of proposed method is totally independent of number of services in a service class (method of computing local constraints is based on extreme values of QoS attributes of service classes). Only the selection phase of the proposed method has to scan the QoS values of individual services, which is inevitable in any method of selection.

Table 21 Comparison results of Experiment 2

# of service classes	# of constraints	# of services	Computation time in ms	
			Alrifai et al. approach	Proposed approach
10–100	3	500	500–20,000	0.029–0.070

Table 23 Computation time with respect to number of service classes—proposed approach versus Lianyong Qi et al. approach

# of service classes	# of constraints	# of services	Computation time in ms	
			Lianyong Qi et al. approach	Proposed approach
5–25	4	50	$10^0 - 10^3$	0.010–0.023

Table 24 Computation time of proposed approach by varying number of services (with experimental conditions similar to Lianyong Qi et al. approach)

# of services	Computation time in ms		
	$t_{decompose}$	t_{select}	$t_{proposed}$
50	0.007	0.003	0.010
100	0.007	0.006	0.013
150	0.007	0.007	0.014
200	0.007	0.009	0.016
250	0.007	0.012	0.019
300	0.007	0.014	0.021
350	0.007	0.015	0.022
400	0.007	0.016	0.023
450	0.007	0.018	0.025
500	0.007	0.019	0.026

4.4.3 *Proposed approach versus Freddy Lecue et al. approach*

In [15], Freddy Lecue et al. have addressed the scalability issue of service composition by selecting composition that satisfies a set of constraints rather than the one that produces optimal utility. The authors modeled service selection as Constraint Satisfaction Problem (CSP) and solved it using stochastic search method. While searching the method uses Hill Climbing algorithm with two functions, namely an evaluation function and an adjacency function. Evaluation function is a function of both quality of semantic link and QoS attributes of services.

Consider a composition c. Let $f(c)$ denote the value of evaluation function of c. The value of $f(c)$ is computed using

$$f(c) = \frac{w_{cd}\,Q_{cd}(c) + w_m\,Q_m(c)}{w_{pr}\,Q_{pr}(c) + w_t\,Q_t(c)} \tag{30}$$

In (30), $Q_{cd}(c)$ denotes the common description rate of c (this factor estimates the rate of descriptions which ensure a correct data flow among the services of the composition), $Q_m(c)$ denotes the matching quality of c (this factor ranges from 0 to 1 based on the standard semantic relations, namely exact, plugin, subsume and intersection), $Q_{pr}(c)$ denotes quality of price of c and $Q_t(c)$ denotes the quality of execution time of c. While searching, the algorithm begins with a random composition c_{final}, and this composition will be replaced by the other combination if the other is adjacent to c_{final}. Here, two compositions are said to adjacent if they differ in exactly one assignment. Reassigning of composition takes place till the algorithm finds a combination that satisfies all constraints.

In Freddy Lecue et al. approach [15], service selection is based on both functional and non-functional characteristics of services. But the proposed work focuses only on the non-functional aspect. Hence, the time taken in selecting services based on non-functional aspect is compared with that of the proposed work. Further, during non-functional selection in order to achieve low computation time, Freddy Lecue et al. approach [15] finds *a single solution* which satisfies all given constraints rather than selecting optimal one from *all solutions*.

Experimental setting similar to Freddy Lecue et al. approach [15] has been brought into the proposed approach for comparison. The number of iterations involved in Freddy

Table 25 Computation time with respect to number of services–proposed approach versus Lianyong Qi et al. approach

# of service classes	# of constraints	# of services	Computation time in ms	
			Lianyong Qi et al. approach	Proposed approach
4	4	50–500	10^0-10^1	0.010–0.026

Table 26 Number of combinations in Freddy Lecue et al. approach and proposed approach

Number of service classes	Number of iterations		Computation time in ms	
	Freddy Lecue et al. approach	Proposed approach	Freddy Lecue et al. approach	Proposed approach
100	8×10^4	35,000	2,912	0.036
200	16×10^7	70,000	4,850	0.084
300	22×10^{11}	105,000	8,142	0.124

Table 27 Computation time of proposed approach versus Freddy Lecue et al. approach with respect to number of services

Number of services	Computation time in ms	
	Freddy Lecue et al. Approach	Proposed approach
280–500	5,000–13,000	0.091–0.104

et al. approach [15] and the proposed approach with respect to number of services classes is given in Table 26. Here, the number of services per service class is taken as 350.

From Table 26, in Freddy Lecue et al. approach [15], the number of iterations required to find the first composition which satisfies the given constraints is found to increase with respect to number of service classes. Experimental results of [15] show that when number of tasks increases from 100 to 200 to 300, the number of iterations is found to increase as 8×10^4, 16×10^7 and 22×10^{11}, whereas in the proposed approach, the number of iterations to be searched is $l \times n$ where 'n' denotes the number of service classes and 'l' denotes the number of services. With 'l' fixed as 350, and when the value of 'n' is increased from 100 to 200 to 300, the number of combinations is found to vary as 35,000, 70,000 and 105,000 which is a linear increase. Correspondingly, from Table 26, the computation time of the proposed approach is found to be very low when compared to Freddy Lecue et al. approach [15].

Further, how the computation time of the methods varies with respect to number of services per service class is given in Table 27. Here, the number of service classes is fixed as 300.

From Table 27, when number of services per task is increased from 280 to 500, the computation time of Freddy Lecue et al. Approach [15] is found to vary from around 5,000 ms to around 13,000 ms, whereas the computation time of the proposed approach varies from 91 to 104 µs. The increase in computation time of the proposed approach with respect to number of service is found to be negligible when compared with Freddy Lecue et al. Approach [15].

Table 28 Time taken to compute QoS for an OR pattern using conventional and proposed methods

Number of sequential paths	Time taken to compute QoS for OR pattern (in μs)	
	Conventional method	Proposed method
2	540	542
3	552	550
4	558	558
5	565	565
6	576	572
7	583	577
8	589	581
9	596	591
10	606	602

4.4.4 Computation time for finding QoS of an OR pattern

Time taken for computing QoS for an OR pattern using conventional method and proposed method is computed by varying the number of sequential paths in OR pattern from 2 to 10 and given in Table 28 and Fig. 25. Each sequential path contains 2 service classes.

From Table 28 and Fig. 25, it is seen that there is no much difference in time taken while computing QoS using the conventional and proposed methods. In both methods, QoS value of each sequential path is obtained by aggregating the QoS values of tasks present in that sequential path. In conventional method, the QoS value of OR pattern is obtained by adding the values which are obtained by multiplying QoS

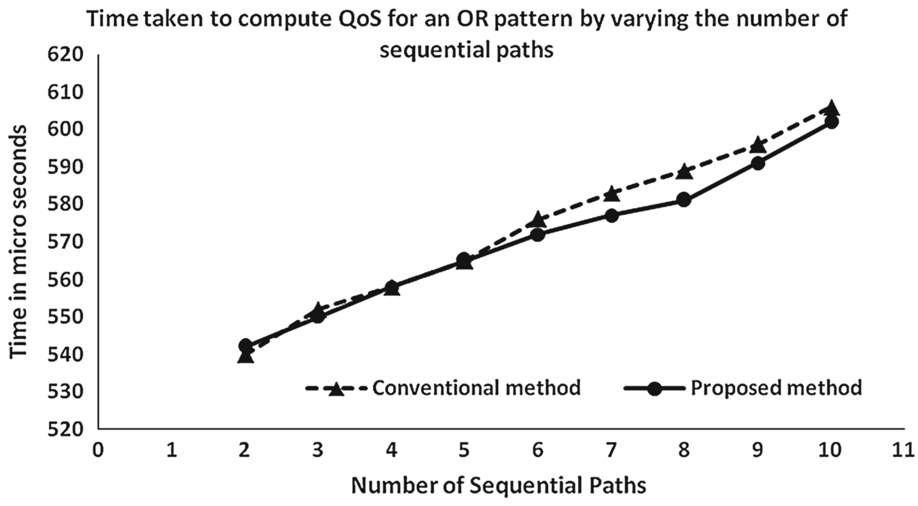

Fig. 25 Time taken to compute QoS for an OR pattern using conventional and proposed methods

Table 29 Typical dataset chosen to illustrate the limitation of local approach

Service ID	Service class-1		Service class-2		Service class-3		Service class-4		Service class-5	
	Response time	Cost	Response time	Cost	Response time	Cost	Response time	Cost	Response time	Cost
1	45	150	61	105	189	400	123	100	200	22
2	234	123	65	145	60	217	100	200	600	150
3	190	99	180	543	45	217	453	234	67	320
4	98	89	56	234	168	218	165	190	100	350
5	450	67	176	213	124	300	170	87	56	100
6	123	560	78	34	99	220	100	78	760	98
7	80	567	167	300	345	300	190	145	344	76
8	350	150	160	95	123	231	456	87	780	76
9	54	70	123	56	160	120	123	90	500	25
10	90	90	40	128	456	218	134	220	1,500	26

of each sequential path by the concerned path's probability, according to (3). In proposed method, the QoS value of the pattern is obtained by finding the maximum of QoS values of all sequential paths according to (4). From Table 28, under the given experimental settings, it is found that there is no much time difference while computing QoS according to (3) and (4) . But the proposed method of computing QoS ensures 100 % guarantee for successful execution to each path of OR pattern when it is given a chance for execution with the constraints are at least equal to minimum time requirement of the pattern, whereas the conventional method fails to provide 100 % guarantee for successful execution.

4.5 Findings

In this work, the problem of QoS-based service selection for real-time applications with combinational workflow having most common execution patterns, AND, OR and Loop is addressed. As the method of decomposing constraints is based on the extreme values of QoS attributes of service classes and the given global constraints, the method becomes independent of number of services in a service class. This feature of independence is unique. The proposed method of decomposing constraints is found to have very low computation time when compared to other approaches. From a series of experiments, it is found that the time characteristics of the method and *utility_ratio* are found to be encouraging with a typical test collection.

In general, any local selection approach reduces the time complexity of global approach by dividing the workflow level selection into a set of task-level selections. Using local constraints, the local selection approach solves the task-level selections quickly. Also, as the aggregation of local constraints is always equal to the global constraints, if a selection satisfies local constraints, it implies that the selection meets the global constraints. But in some situations, a local selection approach fails to detect a feasible solution that may satisfy the given global constraints but not the simultaneous fulfillment of local constraints. In such situations, more than one QoS attribute will be involved. This limitation is illustrated with a typical dataset given in Table 29.

The dataset contains 5 service classes with each service class containing 10 services. Two constraints, namely response time and cost (most commonly used negative attributes), are considered for discussion. The first column of each row denotes service ID. An ith row in Table 29 gives the details of an ith service in different service classes (i.e., from service class-1 to service class-5). The values of response time and cost of all services in different service classes are given in the dataset.

Now, consider a query with QoS constraints "*response time* $<=$ 1000 *and cost* $<=$ 800". For the given QoS constraints, the local constraints of response time and the local constraints of cost for different service classes are computed as per the proposed method, and the values are given in Table 30.

For the above query, the global approach identifies a service combination, consisting of 9th, 10th, 3rd, 6th and 5th services from 1st, 2nd, 3rd, 4th and 5th service classes, respectively, with utility 0.966705 as the best available service combination.

The local approach identifies the best available service from each service class subject to local constraints. It identifies 9th, 10th, 6th and 5th services as the best available services from 1st, 2nd, 4th and 5th service classes, respectively. Here, this approach identifies no best available service from service class-3 as there is no single service in the service class-3 that satisfying both the local constraints of response time and cost simultaneously.

When local approach finds no best available service for a particular service class, then the local constraints of QoS attributes of that service class can be relaxed (provided the

Table 30 Local constraints of response time and cost for different service classes (of the typical dataset)

Service class	Constraint of response time		Constraint of cost	
	Lower bound	Upper bound	Lower bound	Upper bound
1	45	147.92	67	216.61
2	40	59.17	34	207.44
3	45	149.90	120	152.81
4	100	149.90	78	89.39
5	56	493.09	22	133.71

relaxed constraint should satisfy the global constraints) to discover any existing feasible solution. A heuristic approach is suggested for relaxing the local constraints of service class so that efficiency of the local selection approach will be improved. The heuristic approach is illustrated with two constraints, namely response time and cost.

Let Q_k^{global} denote the global constraint of kth attribute. Let $Q_{local}(j,k)$ denote the local constraint of kth attribute of jth service class. Let $Q_k^{assigned}$ denote the sum of kth attribute of all selected best available services. The value of $Q_k^{assigned}$ is computed using (31)

$$Q_k^{assigned} = \sum_{j=1}^{n} q_k(s_{jb}) \,\big|\, s_{jb} \neq null \tag{31}$$

In (31), s_{jb} denotes the best available service of jth service class and $q_k(s_{jb})$ denotes the kth attribute of s_{jb}.

Following are the steps used to relax local constraints of a service class for which the best available service is null. Consider a service class S_j for which s_{jb} is null.

Step 1: Find a set of services from S_j which meet at least one of the given local constraints (i.e., either response time or cost). Let this set of services is denoted by $S_{initial}$.

Step 2: Find all services from $S_{initial}$ which meet the constraint of higher priority attribute. Let these services would form the set $S_{candidate}$. Now, the services in $S_{candidate}$ satisfy the constraints of attribute with higher priority but may not satisfy the constraints of the attribute with lower priority.

Step 3: Consider lth attribute as the low priority attribute. Find the service which has minimum value for lth attribute from $S_{candidate}$. Let this value be $Q_{min}(l)$.

Step 4: Find the remaining or balance value of lth attribute, denoted by $Q_l^{balance}$ after assigning lth attribute to all the best available services using (32). In (32), Q_l^{global} denotes the global constraint of lth attribute and $Q_l^{assigned}$ denote the sum of lth attribute of the best available services of all service classes implementing the workflow (for which $s_{jp} \neq null$).

$$Q_l^{balance} = Q_l^{global} - Q_l^{assigned} \tag{32}$$

Step 5: Let $Q_{local}(j,l)$ denote the local constraint of lth attribute of jth service class. If $Q_l^{balance} > Q_{min}(l)$ relax $Q_{local}(j,l)$ using (33)

$$Q_{local}(j,l) = Q_{\min}(l) \tag{33}$$

When the local constraint of lth attribute is relaxed for jth service class, then the value of $Q_l^{balance}$ should be updated according to

$$Q_l^{balance} = Q_l^{balance} - Q_{local}(j,l) \tag{34}$$

Step 6: After constraints are relaxed, the services which satisfy the simultaneous fulfillment of both the constraints of attribute with higher priority and attribute with lower priority are identified. From the resulting set, the service having the minimum value for higher priority attribute will be chosen as the best available service.

The steps are applied to the above example as follows. For the given query, the local selection approach produces null for the best available service of service class-3. The set $S_{initial}$ is constructed with 2nd, 3rd, 5th, 6th, 8th and 9th services of service class-3. Let us consider response time (first attribute) and cost (second attribute) as the high and low priority attributes, respectively. Now, the set of services that satisfy the constraint of response time are extracted from $S_{initial}$. This set forms $S_{candidate}$ and $S_{candidate}$ contains 2nd, 3rd, 5th, 6th and 8th services from service class-3. There are two services in this set that has minimum value for cost (217). They are second and third services of service class-3. The values of $Q_2^{assigned}$ and $Q_2^{balance}$ are found as 376 and 424, respectively. Here, it is found that $Q_2^{balance} > Q_{\min}(2)$ and hence, the constraint of cost is relaxed from 146 to 217 for service class-3. The value of $Q_2^{balance}$ is updated as 207. Now, both the services (second and third services) in service class-3 satisfy the local constraint of cost. Of these two services, the third service is selected as the best available service for service class-3 as it has minimum value for response time. With this heuristic approach, the best available service combination identified by the local approach is given as 9th, 10th, 3rd, 6th and 5th services from 1st, 2nd, 3rd, 4th and 5th service classes, respectively, with utility score 0.966705.

Thus, when a local approach finds no best available service from a service class, the heuristic approach relaxes the local constraints of such service class and tries to identify feasible service.

Table 31 Extreme values of response time and cost of different service classes

	Service_ class 1	Service_ class2	Service_ class3	Service_ class4	Service_ class5	Aggregated QoS
Minimum_response_time	40.5	36	40.5	90	50.4	257.4
Maximum_response_time	702	280.8	711.36	800	1,800	4, 294.2
Minimum_cost	60.3	30.6	40	70.2	19.8	220.9
Maximum_cost	884.52	847.08	624	365.04	546	3, 266.6

An empirical study has been taken up with the following objectives

- To find the utility for different queries and compare it with that of global approach
- To find the efficiency of the proposed approach and compare it with that of global approach
- To find how heuristics-based constraint relaxation improves the efficiency of the proposed approach
- To find how user's trade-off among various QoS attributes improves the efficiency of the proposed approach.

Here, efficiency is defined as the ratio of number of queries answered within expected time to the number of queries posted within expected time. Efficiency is expressed in %.

$$efficiency = \frac{number\_of\_queries\_answered}{number\_of\_queries\_posted} \times 100 \tag{35}$$

Toward the study, an experiment has been conducted with 5 services classes, each containing 100 services. Response time and cost have been chosen as the interested QoS attributes. The values of response time and cost of all services of different service classes are given in Appendix C (supplementary material) for reference. Consider user's QoS preferences are given as 80 % preference to response time and 20 % preference to cost. Let *grt* and *gc* denote the global constraint of response time and cost, respectively. The availability of appropriate service combination is mainly determined by user's constraints, QoS preferences and the values of QoS attributes of individual services. To find efficiency of different approaches, the QoS values of individual services are kept same for all approaches. The constraints are based on extreme values of QoS of service classes. The extreme values of response time and cost of different service classes are given in Table 31.

Now, the queries are constructed with different QoS constraints that fall between the ranges of aggregated QoS of services. Seventy-five queries are constructed with constraint of response time and constraint of cost ranging from 4,000 to 300. Constraints are decreased in steps of 50. For each query, the utility of the proposed approach is computed and given in Appendix D (supplementary material) for reference. For

Table 32 QoS values of solutions obtained using the proposed and global approaches

Total number of solutions produced by the proposed approach	Number of solutions produced by the proposed approach that have the same QoS values as global approach	Number of solutions produced by the proposed approach that have QoS values different from global approach
66	59	7

Table 33 Average utility obtained using global and proposed approaches

Number of queries	Average utility obtained using global approach (%)	Average utility obtained using the proposed method (%)
66	97.233	97.156

comparison purpose, for each query the utility obtained using global approach is also given in Appendix D (supplementary material). Global method is found to yield answers for constraints in the range 4,000 $\geq grt \geq$ 300 and 4,000 $\geq gc \geq$ 300. The global approach is found to answer 73 queries out of 75 queries, and its efficiency is found to be 97.3 %. The proposed approach is found to answer 66 queries. The QoS values and utility values obtained for these 66 queries have been compared to the QoS values and utility values obtained using global approach as given in Tables 32 and 33.

From experimentation, the proposed approach is found to answer the queries for the constraints in the range, 4000 $\geq grt \geq$ 750 and 4000 $\geq gc \geq$ 750. When the user's constraints are very close to the minimum values of QoS attributes, simultaneous fulfillment of local constraints may not get fulfilled. In our example, when 700 $\geq grt \geq$ 300 and 700 $\geq gc \geq$ 300, the proposed local approach fails to meet the simultaneous fulfillment of local constraints. Now, the cost constraint is relaxed using heuristics method of constraint relaxation subject to the condition that the aggregation of local constraints of cost always satisfies the given global constraint of cost. When the proposed approach is combined with this heuristics-based constraint relaxation, feasible solutions are obtained for constraints in the range

Table 34 Number of queries answered and efficiency of different approaches

	Proposed local selection	Proposed local selection with heuristics-based constraint relaxation	Proposed local selection with trade-off
No of queries answered	66	5	4
Efficiency	88 %	95 %	100 %

$700 \geq grt > 450$ and $700 \geq gc > 450$. This kind of constraint relaxation may not find a feasible solution when relaxation fails to satisfy the global constraint. In this example, when $450 \geq grt \geq 300$ and $450 \geq gc \geq 300$, local approach in combination with heuristics fails to find a solution, because constraints cannot be relaxed beyond a particular condition as the relaxed constraints violate the global constraints. In this range, user's trade-off among QoS attributes are considered. When we consider user's trade-off among various attributes, all QoS constraints will not be fulfilled. In this example, two constraints are considered. Consider that the constraint of response time is given higher priority, and it is a hard constraint (which much be definitely satisfied) than the constraint of cost, a soft constraint (which may or may not be satisfied). While selecting services, if no solution is found, then the constraint having lower priority (i.e., cost) will be compromised and ultimately the cost constraint may not be met. In this example, when $450 \geq grt \geq 300$ and $450 \geq gc \geq 300$, local approach in combination with user's trade-off helps in finding a feasible solution. The number of queries answered by (i) the proposed approach, (ii) proposed approach in combination with heuristics-based constraint relaxation (when proposed method fails) and (iii) the proposed method in combination with user's trade-off when both proposed approach and proposed in combination with heuristics fail are computed. The efficiency of the above approaches is calculated using (35). The number of queries answered by different approaches and the efficiency of the approaches are given in Table 34. The efficiency of the above approaches is given in Fig. 26.

From Table 34 and Fig. 26, the efficiency of the proposed method is found to be 88 %. When it is combined with heuristics-based constraint relaxation, its efficiency is found to increase by 7 %. Further, when both proposed approach and proposed approach with heuristics fail, user's trade-off improves the efficiency of the approach by 5 %. Also, while finding best services, there may be little probability to obtain more than one service combinations as best service combination. This can occur when more than one service combination produces the same utility. If at all more than one service combinations are obtained as best combinations, one of the combinations will be chosen for composition while others can be considered as alternatives to the chosen one which may fail to deliver the expected task due to runtime errors.

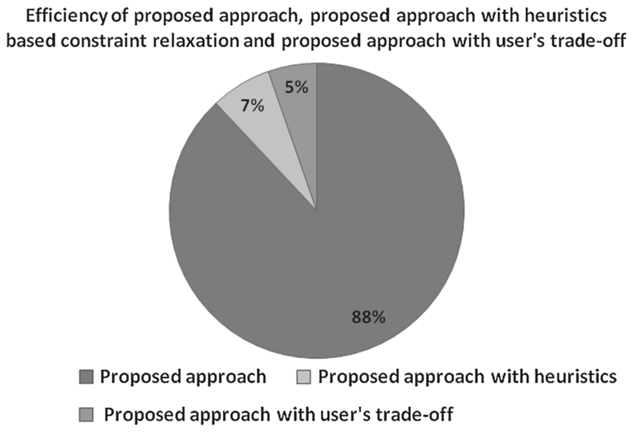

Fig. 26 Efficiency of proposed method, proposed method with heuristics and proposed method with user's trade-off

5 Conclusion

This paper presents a newly developed local selection methodology for selecting best available service combination for a given workflow having most common business structures AND, OR and Loop based on QoS. The methodology has been implemented and tested by a series of experiments. This methodology is found to yield excellent time characteristics and *utility_ratio* when compared with existing approaches. This paper describes a new method for computing response time of OR execution pattern, which guarantees 100 % successful execution of every path in an OR unit when the path is given a chance for execution. But the existing approaches assign response time to an OR unit based on *'expected response time'* of an OR unit in which all the paths of an OR unit are not guaranteed for successful execution when a chance is given for execution. In this case, even a workflow may fail to get executed when any one of its OR path with insufficient time (assigned as per expected response time) gets a chance to execute. The proposed methodology alleviates this shortcoming.

From experiments, it is found that the retrieval of QoS values of services from its storage/repository to a concerned service application is found to consume time of the order of few tens of milliseconds, and it is recommended that the retrieval of QoS must be done prior to querying for quick selection. The method identifies a service combination which is the best among the available feasible combinations rather than identifying optimal combination, which is not certain to exist in real situations. Further, the methodology suggests a heuristic approach to relax local constraints and to improve the efficiency of local selection approach. The experimental evaluations show significant improvement in computation time, *utility_ratio* and detection efficiency while identifying the best available services for composition. This is especially useful for applications with real-time composition.

References

1. Alrifai M, Risse T (2008) Efficient QoS-aware service composition. In: The proceedings of the 3rd workshop on emerging web services technology. IEEE, Los Alamitos, CA, pp 60–70
2. Alrifai M, Risse T (2009) Combining global optimization with local selection for efficient QoS-aware service composition. In: The proceedings of the 18th international conference on World Wide Web, ACM, pp 881–890
3. Alrifai M, Risse T, Dolog P, Nejdl W (2009) A scalable approach for QoS-based web service selection. In: Service-oriented computing, 2008 workshops. Springer, Berlin, pp 190–199
4. Alrifai M, Skoutas D, Risse T (2010) Selecting skyline services for QoS-based web service composition. In: The proceedings of the 19th international world wide web conference. ACM, North Carolina, USA, pp 11–20
5. Anselmi J, Ardagna D, Cremonesi P (2007) A QoS-based selection approach of autonomic grid services. In: The proceedings of the workshop on service-oriented computing performance: aspects, issues and approaches. ACM Press, pp 1–8
6. Ardagna D, Pernici B (2007) Adaptive service composition in flexible processes. IEEE Trans Softw Eng 33(6):369–384
7. Ardagna D, Pernici B (2005) Global and local QoS constraints guarantee in web service selection. In: The proceedings of IEEE international conference on web services. (FL, USA). IEEE Computer Society, pp 805–806
8. Canora G, Esposito R (2004) A lightweight approach for QoS-aware service composition. In: Proceedings of 2nd international conference on service oriented computing. New York, pp 37–46
9. Cardoso J, Sheth AP, Miler JA, Arnold J, Kochut K (2004) Quality of service for workflows and web service processes. J Web Semant 1(3):281–308
10. Chen Z, Wang H, Pan P (2010) An approach to optimal web service composition based on QoS and user preferences. In: International joint conference on artificial intelligence, IEEE, Jinan, China, pp 96–101
11. Gao Y, Zhang B, Na J, Yang L, Dai Y, Gong Q (2006) Optimal selection of web services with end-to-end constraints. In: The proceedings of first international multi-symposiums on computer and computational sciences, IEEE, China, pp 460–467
12. Hong L, Hu J (2009) A multi-dimension QoS based local service selection model for service composition. J Netw, 4(5), Academy Publisher, pp 351–358
13. Jin J, Cao Y, Zhu D, Pu X, Yang M (2010) A structure-wise service selection approach for efficient service composition. In: IEEE International conference on E-business engineering (Beijing, China), pp 256–261
14. Jin J, Zhang Y, Cao Y, Zhou R (2010) An enhanced QoS decomposition approach for efficient service composition. In: The proceedings of the fifth IEEE international conference on computer science & education, (Beijing, China), pp 1680–1684
15. Lecue F, Mehandijiev N (2009) Towards scalability of quality driven semantic web service composition. In: IEEE International conference on web services, IEEE Computer Society, Los Angeles, USA, pp 469–476
16. Li W-J, Li X, Liang X-J, Zhou X-C (2011) QoS-driven service composition with multiple flow structures, In: The proceedings of IEEE international conference on services computing, IEEE Computer Society, Washington DC, USA, pp 362–369
17. Liangzhao Zeng, Boualem Benatallah (2004) A QoS-aware middleware for web service composition. IEEE Trans Softw Eng 30(5):311–327
18. Li J, Zhao Y, Liu M, Sun H, Ma D (2010) An adaptive heuristic approach for distributed QoS-based service composition. In: IEEE symposium on computers and communications, Beijing, China, pp 687–694
19. Menasce DA (2004) Composing web services: a QoS view. IEEE Internet Comput 8(6):88–90
20. Oster ZJ, Santhanam GR, Basu S (2011) Identifying optimal composite services by decomposing the service composition problem. IEEE international conference on web services, (USA), pp 267–274
21. Qi L, Tang Y, Dou W, Chen J (2010) Combining local optimization and enumeration for QoS-aware web service composition. In: IEEE international conference on web services, IEEE Computer Society, pp 34–41
22. Senivongse T, Wongsawangpanich N (2011) Composing services of different granularity and varying QoS using genetic algorithm. In: The proceedings of World Congress on engineering and computer science, International Association of Engineers, San Francisco, USA, vol. I, pp 388–393
23. Sherry SX, Zhao J, Wang H, Winter R, Zhao JL, Aier S (2010) A negotiation based approach for service composition. In: DESRIST 2010: LNCS 6105, Springer, Berlin, pp 381–393
24. Xiong PC, Fan YS, Zhou MC (2008) QoS-aware web service configuration. IEEE Trans Syst Man Cybern: Part A 38(4):888–895
25. Yanwei Z, Hong N, Haojiang D, Lei L (2010) A dynamic web services selection based on decomposition of global QoS constraints. In: The proceedings of IEEE youth conference on information computing and telecommunications. Beijing, China
26. Yoon KP, Hwang CL (1995) Multiple attribute decision making: an introduction (Quantitative Applications in the Social Sciences). Sage Publications
27. Yuan-sheng L, Zhen-Hong T, Lu-Lu Y, Hong-Tao X, Zhi-hong X, Zhi-feng W (2010) A QoS-based web service dynamic composition framework. In: The proceedings of 9th international symposium on distributed computing and applications to business, engineering and science, (Hong Kong), pp 188–192
28. Yu T, Lin KJ (2005) A broker-based framework for QoS-aware web service composition. In: IEEE international conference on eTechnology eCommerce and eService, IEEE, Hong Kong, pp 22–29
29. Yu T, Zhang Y, Lin K-J (2007) Efficient algorithms for web services selection with end-to-end QoS constraints. ACM Transactions on Web, ACM publication, Vol. 1, No. 1, Article 6
30. Zeng L, Benatallah B, Dumas M, Kalagnanam J, Sheng QZ (2003) Quality driven web services composition. In: The proceedings of the 12th international conference on World Wide Web, ACM Press, Hungary, pp 411–421
31. Zhang W, Carl CK, Feng T, Jiang H-Y (2010) QoS-based dynamic web service composition with ant colony optimization. In: 34th annual IEEE computer software and applications conference. Seoul, Korea, pp 493–502