This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589244, IEEE Transactions on Services Computing

QoS Dependency Modeling for Composite Systems

Khayyam Hashmi, Zaki Malik, Abdelkarim Erradi, Abdelmounaam Rezgui

Abstract—Software as a service is a well accepted software deployment and distribution model that has grown exponentially in the last few years. One of the biggest benefits of SaaS is the automated composition of different services in a composite system. It allows users to automatically find and bind these services (to maximize the productivity), meeting both functional and non-functional requirements. In this paper, we present a framework for modeling the dependency relationships among different Quality of Service parameters of the component services. Our proposed approach considers the different invocation patterns of component services, and presents a service composition framework to model the dependencies and uses the global QoS for service selection. We evaluate the efficiency of our proposed technique on the WSDream-QoS Dataset [1].

Index Terms—QoS, Web Service, Dependency Modeling.

1 Introduction

In recent years, the Service-Oriented Architecture (SOA) paradigm has gained momentum as a means to develop applications. In SOAs, loosely-coupled software artifacts (commonly referred to as services) may implement specialized functionalities which can be combined with other services from various business partners or public entities into composite services to provide some value-added functionality. Two major entities are involved in any SOA transaction: Service consumers, and Service providers. As the name implies, service providers provide a service on the network with the corresponding service description [2]. A service consumer needs to discover a service to perform its desired task among all the services published by the different providers. The consumer binds to the newly discovered service(s) for execution, where input parameters are sent to the service provider and output is returned to the consumer. In situations where a single service does not suffice, multiple services can form a composite system to deliver the required functionality [3].

In a running composite system, services may exhibit errors, undergo changes, or become unavailable, and may need to be replaced by other services (to achieve better performance or lower cost). The process of composing/replacing service(s) is usually time consuming, error-prone and often non-optimal. Using automated composition techniques one could improve upon these results. Apart from the functional properties, within SOAs, Quality of Service (QoS) attributes express the non-functional aspects of a service, such as response time, cost, reliability or the supported security protocols etc. The overall performance of a composite system thus depends heavily on how the individual QoS values of the component services effect the composite solution. The orchestration of individual services in a composition also plays a significant role on the QoS values of the composition. Most existing approaches consider this as a local (service level) optimization problem and lack a coherent framework for the specification, and optimization of service compositions focusing on the global (system-wide) QoS properties of the system considering the orchestration and dependency relationships among component services.

Services in SOAs are *autonomous*, i.e., they are independently deployed, and the topology is *dynamic* where a service can leave the system, or fail without notification. In any SOA the various component services may be invoked using a different invocation model. Here, an invocation refers to *triggering a service (by calling the desired function and providing inputs) and receiving the response (return values if any) from the triggered service*. There are six major invocation relations defined in the literature for service compositions [4] [5]: Sequential Invocation, Parallel Invocation, Probabilistic Invocation, Circular Invocation, Synchronous Activation, and Asynchronous Activation. A brief overview of these follows.

Sequential Invocation: In sequential invocation, a service S invokes a service A. It is denoted as Sequential (S : A) (see Figure 1-(a)). Sequential invocation is also defined as a serial invocation.

Parallel Invocation: In parallel invocation, a service S simultaneously invokes multiple services. For example, if S has service A and service B which are independent and successors of S, S can invoke both A and B at the same time. It is denoted as Parallel (S : A, B) (see

- *K. Hashmi and Z. Malik are with the Department of Computer Science, Wayne State University, Detroit, MI, 48202.*
E-mail: khayyam, zaki@wayne.edu
- *A. Erradi is with Department of Computer Science and Engineering (CSE), Qatar University, Al Tarfa, Doha 2713, Qatar.*
E-mail: erradi@qu.edu.qa
- *A. Rezgui is with Department of Computer Science and Engineering, New Mexico Tech, NM.*
E-mail: rezgui@cs.nmt.edu

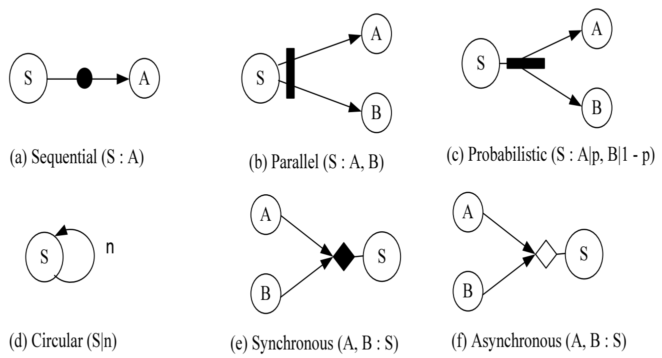

Fig. 1. Major SOA Invocation Models

Figure 1-(b)).

Probabilistic Invocation: In probabilistic invocation, a service S invokes service(s) with a probability (see Figure 1-(c)). For example, if S invokes service A with the probability p and service B with the probability 1 - p, it is denoted as Probabilistic (S : A|p, B|1 - p). The probabilistic invocation is also defined as fork invocation.

Circular Invocation: In circular invocation, a service S invokes itself multiple (n) times. It is denoted as Circular (S|n). A circular invocation can thus be defined as sequentially invoking itself n times (see Figure 1-(d)).

Synchronous Activation: In synchronous activation, a service S is invoked only after all of its preceding services that were invoked in parallel have completed their execution. For example, if S has synchronous preceding (invoked in parallel) services A and B, both these services would need to complete before S can progress. It is denoted as Synchronous (A, B : S) (see Figure 1-(e)).

Asynchronous Activation: In asynchronous activation, a service S is invoked only after one of its preceding services that were invoked in parallel have completed their execution. For example, if S has asynchronous preceding (invoked in parallel) services A and B, either A or B's completion would cause S to progress. It is denoted as Asynchronous (A, B ; S) (see Figure 1-(f)).

It is highly likely that a composite system exhibits more than one type of service invocation. Since certain system properties are a composite function of its component services (e.g. the overall throughput in transactions per second of the system is dependent on the component service that has the least transaction per second - bottleneck), it is important to consider these invocation patterns while considering and modeling the dependencies of the different QoS attributes of the component services. In this paper, we propose a service composition approach that attempts to mitigate the fact that different users may experience different QoS values for a given service, and hence when the service is composed into a system, the observed QoS values could be different from the published QoS values. We use a collaborative filtering based approach to predict the observed QoS values for a service, then use the composition structure to model the dependency modeling relationship among different QoS values of component service, and then use a multi-objective Markov Decision Model to formulate a globally optimal solution.

The rest of the paper is organized as follows. Section 2 presents our motivating example. Section 3 presents our dependency modeling framework. Section 4 discusses the collaborative filtering approach. Section 5 details our multi-objective service composition approach. In Section 6 we present the results of our experiments and Section 6 discusses the literature review. Section 8 presents the conclusion and our future directions.

2 Service Composition Optimization Strategies

In this section we look at different service composition optimization goals, and the quantity of information available for the decision making process. It is assumed that all the services under discussion are functionally equivalent and QoS attributes are the primary selection parameters. We start by defining a travel reservation service example to motivate the problem and our proposed approach.

2.1 Sample Scenario

Assume that a user wants to attend a conference and needs to make travel arrangement for his journey. He needs to purchase an airline `ticket` and reserve a `hotel` for this travel. Moreover, he needs some `transportation` to go from the airport to the hotel and from the hotel to other venues "Site-seeing". Assume that the user would be using a SOA-based online service that is a *one-stop shop* providing all the five options (airline ticket, hotel, attractions and transportation) through different component services.

The online reservation system provides many services such as: attraction service which are outsourced to three services (representing individual services): Art, Museums, and Area tours. This service provides arrangement to visit different areas through sub-contractor companies. For clarity, Figure 2 shows the different options. User may select Art, Art and Museum, Art and Area tour, Museum or Area tour. The transportation service also works through subcontractors that provide car, bus or bike. These companies provide different services based on the distance between the places user plans to visit (user has the option to choose the mode of transportation). The system also provides services to calculate the distance between two points. The system employs two other services: one for airline ticket, and the other for hotel reservation.

Figure 2 shows a sample structure of the travel reservation scenario. This structure depicts a combination of different invocation models that may be used to compose the system. Since the user is looking for a travel arrangement that include: booking a ticket, booking a hotel,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589244, IEEE Transactions on Services Computing

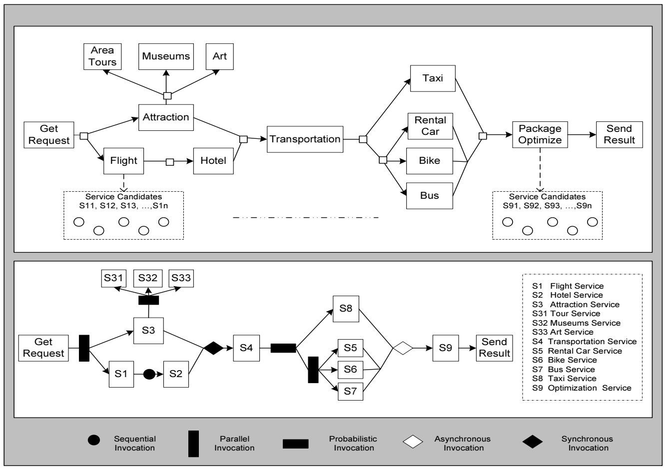

Fig. 2. Scenario with Invocation Models

transportation (car, bike or bus) based on the distance between the places and visiting some attractive places, these services can be invoked in parallel. Booking a ticket and find attractions is an example of *parallel invocation*. There are three choices that provide attraction (Area tour, Museums and Art). Since user has to choose among these service instances, this is an example of *probabilistic invocation*. Similarly car, bike and bus services can be classified as probabilistic invocations. When the system provides the results of transportation to the reservation service, for generating a discount, this invocation is an example of *asynchronous invocation*. Finally, *synchronous invocation* appears when *package optimization* service waits for other services (hotel, ticket and attraction) to compute the final trip cost.

2.2 Local Optimization

The most basic scenario for service selection deals with the local resources and constraints. In the local optimization approach, the selection of a component Web service that performs a given task of the composite system is done as a stand alone component without considering any system-wide constraints or limitation of other components of the composition. When the composite system wants to perform a certain task (as a part of initial system composition or replacing a faulty service during execution process) the system gathers QoS information for each candidate Web service. After collecting the QoS information, the system constructs a QoS vector which is then evaluated to calculate the usefulness of each service. The candidate services are then ranked based on their usefulness or utility and the best available service is selected. This selection process could include user defined weights and constraints such as execution time, availability etc.

In our running scenario from Figure 2 assume that the system is considering candidates S11, S12 and S13 for the `flight` service. Assume that the QoS vector used to make the decision consists of four QoS attributes *<Throughput, Availability, Reliability, Execution Time>*. For the sake of simplicity, we assume that the service prices are static and all the above QoS attributes are equally important for the system. Assuming that the QoS vector for S11 is <91,95,95,88>, for S12 it is <94,92,92,88> and S13's vector is <90,98,98,90>. For local optimization the system will select service S13. Although S11 has a lower value for execution time but S13 has much higher values for Availability and Reliability. The system is able to make this decision since it has all the local information regarding the candidate services to compare and make an informed decision. The selected service was the best possible choice in the local context i.e. local maxima.

2.3 Global Optimization

Since local optimization is performed on a per service basis, it is possible that it may not be the optimal choice in the context of the composite system. In Figure 2 let us assume that `hotel` service has a QoS vector of S2 < 93,95,95,92>. One of the system wide constraints could be that both `flight` and `hotel` service should finish their execution in 180 ms. In order to meet this constraint, the system cannot choose service S13 <90,98,98,90> since it does not satisfy the system constraint (i.e. 92 + 90 = 182 > 180). Hence, it would have to select S11 <91,95,95,88> which is clearly not the best local choice but turns out to be the best global choice for the composite system. Global optimization is thus heavily dependent on the amount of system wide information and constraints for optimal decision making. Similarly, if we add another dimension of information, e.g. that we are looking to optimize the throughput (after meeting the execution time constraint) then we can see that service S12 <94,92,92,88> will be a better choice since it has a higher throughput value; although it was the second best choice in the previous scenario, and the last choice in local optimization.

Let us look at another case of global optimization in our running example by adding structural information to our current discussion of service selection. Assume that we have two options for `Attraction` service: S3a <98,95,95,180 > and S3b <93,96,96,180>. Here service S3a is a better choice for `Attraction` service since it has far better throughput and comparable values of other QoS components. Now if we look at the structure of our system in Figure 2 we see that `Attraction` service is invoked in parallel with `Flight` and `Hotel` services which in turn, are sequential in nature i.e. Parallel (Get Request: S3, Sequential (S1:S2)). We can see that based on our current selection of S2 < 93,95,95,92> for `hotel` and S12 <94,92,92,88> for `flight` the maximum throughput for the current composite solution with the Sequential invocation (S12:S2) is 93 (min (93,94)). This, in turn implies that the maximum throughput of the current composite solution with the Parallel invocation (Get Request: S3, Sequential (S12:S2)) cannot exceed 93 i.e. (min (98,93)). Based on this information our choice of S3a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589244, IEEE Transactions on Services Computing

<98,95,95,180 > for `hotel` service is not a global optimal service for the composite system as S3b <93,96,96,180> offers better QoS values for availability and reliability while still matching the maximum throughput value of 93 for the parallel invocation of the scenario. Hence, the service selection framework could benefit from the invocation pattern information for making optimal decisions to improve the overall QoS of the composition. To differentiate, we refer to such relationships among the same QoS components of different service of the composition as *dependency modeling* in the rest of the paper.

3 Dependency Modeling Framework

In order to demonstrate our proposed approach for QoS dependency modeling we will be using the following quality components (note that other QoS attributes can be modeled in a similar manner).

- *Cost* The cost $q_{co}(s, f_i))$ of an operation f_i of a service s is cost incurred (fee paid to the service provider) to invoke an operation in order to perform a particular task. These values are either published by the service providers or are available on demand. The cost is measured as a per invocation unit.
- *Execution Time* Given an operation f_i of service s, the execution time $q_{et}(s, f_i)$ measures the difference between the time when the request is sent and the time when the results are received. The total execution time is the sum of the processing time $T_{process}(s, f_i)$ and the transmission time $T_{trans}(s, f_i)$. It is measured in milliseconds.
- *Reliability* The reliability $q_{re}(s)$ of a service s is measured as the probability of a system to accurately respond to a request (i.e., the operation is completed and a message indicating that the execution has been successfully completed is received by a service requestor) within the maximum expected time frame indicated in the Web service description. The reliability (or success rate) is dependent on the hardware and/or software configuration of Web services and the network connections between the service requesters and providers. The value of the reliability can be computed from data of past invocations using the expression $q_{re}(s) = (K - N_f(s))/K$, where $N_f(s)$ is the number of faulty executions of the service s, and K is the number of times service s has been invoked.
- *Availability* The availability $q_{av}(s)$ of a service s is the probability that the service is accessible. The value of the availability of a service s is computed using the expression $q_{av}(s) = T_a(s)/\theta$, where T_a is the total amount of time for which service s is available over and observed θ amount of time. Availability is usually measured in the percentage. The higher the value is, the more a system is available.
- *Throughput* The throughput $q_{th(s)}$ of a service s is measured as the number of completed requests per unit amount of time.

Table 1 shows how to calculate different QoS values for major SOA invocation models as discussed above.

For the composite system we would need to calculate the over all QoS values suitable for negotiation for the new requested service. This is mainly dominated by the structure of the composition. Apart from the simple invocation methods mentioned above we also need to look into complex invocation patterns. One of the key elements of a composite system is a complex loop. A complex loop can be defined as a *Circular Invocation* with a linear or non-linear complex execution path. Loops may contain different combinations of *Invocation Patterns* i.e. nested loops, probabilistic invocations etc. Hence, the QoS values of a complex loop structure could be calculated by the probability of number of iterations of the loop i.e. p_c, the probability of the exiting the loop p_e and QoS values of the execution path of the loop. We can always flatten a loop structure into repeatable blocks of linearly executing patterns. A loop may have multiple entry and exit points. We can assume that the actual entry and exit points of a loop structure could be ignored for the ease of calculation as they have minimal affect on the actual QoS values. However, the probability of exiting out of the loop p_e along with the individual probabilities of the different execution patterns within the loop are required for the QoS calculations.

Let us assume that the transition probability of execution of Service s_{i+1} after executing Service s_i is denoted by p_i (in a linear execution path this will always be equal to 1). The probability of exiting a loop after executing Service s_i is denoted by $p_{e_{i,j}}$ (in a linear execution path this will be 0) where $p_i + \sum_{j=1}^{m} p_{e_{i,j}} = 1$ and $i \in [1, n]$. The cost of the loop is depicted by c_i and the total execution time is shown as t_i. After x executions of the loop where $x \in [0, +\infty]$ the loop can been transformed into x linear executions paths for the component services and we already know that the probability of executing the next service is $p_{e_{i,j}}$ where $j \in [1, m]$. So the probability for the combination of service for the execution for x where $x \in [0, +\infty]$ time and the execution of any service after Service S_1 is $(\prod_{i=1}^{n} pi)^x p_{e_{1,j}} (x \in [0, +\infty], j \in [1, m])$. Similarly the probability for executing Service S_{k+1} after executing Service S_k where $k \in [2, n]$ is $(\prod_{i=1}^{n} pi)^x (\prod_{i=1}^{k-1} pi) p_{e_{1,j}} (x \in [0, +\infty], j \in [1, m])$. Let $p_0 = 1$ then the probability of executing a service after Service S_k is

$$p'_{e_{k,j}} = \sum_{x=0}^{+\infty} (\prod_{i=1}^{n} p_i)^x (\prod_{i=0}^{k-1} p_i) p_{e_{k,j}} \quad (1)$$

$$= \frac{(\prod_{i=0}^{k-1} p_i) p_{e_{k,j}}}{1 - \prod_{i=1}^{n} p_i} where (k \in [1, n], j \in [1, m]) \quad (2)$$

TABLE 1
QoS values for different invocation models

	Sequential	Probabilistic	Circular	Parallel Synchronous	Parallel Asynchronous
Cost	$\sum_{i=1}^{n} q_{co}(f_i)$	$\sum_{i=1}^{n} p_i * q_{co}(f_i)$	$q_{ac}(f) * c$	$\sum_{i=1}^{n} q_{co}(f_i)$	$\sum_{i=1}^{n} q_{co}(f_i)$
Execution Time	$q_{et}(s, f_i) = T_{process}(s, f_i) + T_{trans}(s, f_i)$	$\sum_{i=1}^{n} q_{et}(s, f_i)$	$q_{ac}(s, f_i) * c$	$max(q_{et}(s_1, f_i), ..., q_{et}(s_n, f_i)))$	$max(q_{et}(s_1, f_i), q_{et}(s_2, f_i), ..., q_{et}(s_n, f_i)))$
Reliability	$\prod_{i=1}^{n} q_{re}(f_i)$	$\sum_{i=1}^{n} p_i * q_{re}(f_i)$	$q_{re}(f)^c$	$\prod_{i=1}^{n} q_{re}(f_i)$	$max(q_{re}(f_1), q_{re}(f_2), ..., q_{re}(f_n)))$
Availability	$\prod_{i=1}^{n} q_{av}(f_i)$	$\sum_{i=1}^{n} p_i * q_{av}(f_i)$	$q_{av}(f)^c$	$\prod_{i=1}^{n} q_{av}(f_i)$	$max(q_{av}(f_1), q_{av}(f_2), ..., q_{av}(f_n)))$
Throughput	$min(q_{th(s_1)}, q_{th(s_2)}, ..., q_{th(s_n)})$	$\sum_{i=1}^{n} p_i * q_{th}(s, f_i)$	$q_{th}(s)$	$min(q_{th(s_1)}, q_{th(s_2)}, ..., q_{th(s_n)})$	$max(q_{th(s_1)}, q_{th(s_2)}, ..., q_{th(s_n)})$

Now the probability that service $S_1...S_n$ is executed x times and then the loop is terminated at service S_k is $(\prod_{i=1}^{n} p_i)^x(\prod_{i=0}^{k-1} pi)(1-p_k)$. The cost for this execution accumulated by the services will be $x\sum_{i=1}^{n} c_i + \sum_{i=1}^{k} c_{i'}$ and the execution time will be $x\sum_{i=1}^{n} t_i + \sum_{i=1}^{k} t_{i'}$. The reliability for this execution will be $(\prod_{i=1}^{n} q_{re(i)})^x(\prod_{i=1}^{k} q_{re(i)})$. Hence we get the following equations for Cost (q_{co}), Total Execution Time (q_{et}), Reliability (q_{re}) and Availability (q_{av}) respectively.

$$q_{co} = \sum_{k=1}^{n}\sum_{x=0}^{+\infty}\left((\prod_{i=1}^{n} p_i)^x(\prod_{i=0}^{k-1} p_i)(1-p_k)(x\sum_{i=1}^{n} c_i + \sum_{i=1}^{k} c_i)\right) \tag{3}$$

$$q_{et} = \sum_{k=1}^{n} \frac{(\prod_{i=0}^{k-1} p_i)(1-p_k)(\sum_{i=1}^{k} t_i + \prod_{i=1}^{n} p_i \sum_{i=k+1}^{n} t_i)}{(1-\prod_{i=1}^{n} p_i)^2} \tag{4}$$

$$q_{re} = \sum_{k=1}^{n}\sum_{x=0}^{+\infty}\left((\prod_{i=1}^{n} p_i)^x(\prod_{i=0}^{k-1} p_i)(1-p_k)(\prod_{i=1}^{n} r_i)^x \prod_{i=1}^{k} r_i)\right) \tag{5}$$

$$q_{av} = \sum_{k=1}^{n} \frac{(\prod_{i=0}^{k-1} p_i)(1-p_k)\prod_{i=1}^{k} r_i}{1-\prod_{i=1}^{n}(p_i * r_i)} \tag{6}$$

$$q_{th} = \sum_{k=1}^{n} \frac{(\prod_{i=0}^{k-1} p_i)(1-p_k)\prod_{i=1}^{k} r_i}{1-\prod_{i=1}^{n}(p_i * r_i)} \tag{7}$$

After calculating the individual QoS values for each invocation pattern we now need to calculate the max possible QoS value for each component of the QoS vector in order to establish a target QoS value that would be used for global optimization. Since our QoS calculation follow a single entry and single exit format we can use the refined process tree approach [6] to find the different execution paths and apply our previously described invocation patterns. In this regard, Algorithm 1 calculates the individual QoS values for all components of the QoS vector at a given node. In this algorithm we use depth first on all the nodes and create tree for all the path to the leaf nodes. Once we get to the leaf node we recursively backtrack and calculate the QoS value for that path. If we run into a conditional branch we add both the paths as child node and this allows us to still find all the possible execution paths for the composition. We need to focus on the QoS value for the paths that contain the service that we want to replace. This will ensure that for global

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589244, IEEE Transactions on Services Computing

optimization we are only considering the relevant paths. Initially, we will set the QoS values for service to be replaced at the maximum possible values to calculate the target QoS negotiation values for the service (refer to section 2). We can see that the proposed depth first algorithm has low time and space complexity. With N nodes and E edges the first exploration of the graph takes $O(|V|+|E|)$ and in the second exploration for the graph we only visit all nodes once, (since the QoS calculation at every invocation point takes a linear amount of time) our execution takes $O(|V|+|E|)$ time.

Algorithm 1 QoS Calculation for a service execution

```
DepthFirstQoS ( Node N, Tree F )
T = create a new tree;
Add T as child of F;
if N has no children then
    Get a path to parent node of N
    for All component of QoS vector do
        Apply the QoS pattern and multiply it with the probability to get the QoS value at current invocation point.
    end for
end if
for each child node of N do
    if isVisited = true then
        Add all the child nodes to T
        if current node = leaf node then
            Get a path to parent node of T
            for All component of QoS vector do
                Apply the QoS pattern and multiply it with the probability to get the QoS value at current invocation point.
            end for
        else
            Call DepthFirstQoS(current node, T)
        end if
    end if
end for
```

Once we determine the dependency relationship we need to be able to predict the QoS values that we may be able to get from our composition. That could be done by predicting the observed QoS values of individual services and then using them in the dependency calculation. We present our collaborative filtering approach in the next section that is used for this purpose.

4 Collaborative Filtering

In ideal situation Web services should meet all the published performance criteria for all of its users. However it is rarely the case. We use a collaborative filtering based approach to mitigate the discrepancies between published and observed QoS values for the Web services. We can loosely classify the Web service attributes into two broad classes of certain and uncertain attributes. The value of certain attributes remains fixed or unvarying for all the users i.e. service name, service provider, cost (publisehd/mutually agreed upon). The value of uncertain attributes on the other hand may change or vary from user to user, or even for the same user, from one invocation to the other invocation, mostly based on the environmental factors. These attributes have a marked tendency to deviate from their published values. Response time is one such example of an uncertain attribute. Many factors can influence the response time of a service e.g. internet speed, network congestion, slow hardware. These factors are usually out of the hands of the service provider. These uncertain attributes may influence the perceived overall QoS value of the systems. However, most of the service selection algorithms do not take the uncertainty of these attributes into account which leads to inaccurate/sub-optimal service matching, these system only consider the published QoS values of the services. However, as discussed above the observed QoS value may differ from the published QoS values for a user. It is well accepted that it is impractical (and nearly impossible) for a service consumer to invoke all the services (under consideration) to record their observed QoS values and then use them for service composition. We can instead utilize the experience of other (similar) users to predict the observed QoS values for services and use them as a metric for service composition. This requires us to investigate an approach that first of all would be able to search users that are similar (in terms of environment) to the current service consumer and have invoked the service under consideration with reported observed QoS values for the service. Secondly, use the experience of similar users to predict observed QoS values for the service consumer. These predicted QoS values could later be used in the negotiation process to minimize the affect of uncertainty of QoS values of the services. Note that it is common for the end user to receive lower values of QoS components than the promised/published QoS values (observed response time of 98ms as compared the published response time of 60ms) and very rare if not highly improbable to experience better than published QoS values. Hence, it is safe to assume that using observed QoS values will not over compensate a service provider for its published QoS values which in turn, could result in over promise of QoS to the consumer.

We use collaborative filtering on the observed QoS values to solve the problem of uncertainty. The two most commonly used memory based collaborative filtering approaches are Pearson correlation and Vector cosine based similarity. Pearson coefficient [7] is symmetric, invariant to scale and location of variable and can also detect negative correlation making it suitable for our problem at hand. The similarity between two users i and j can be calculated using Pearson Correlation Coefficient as:

$$sim(i,j) = \frac{\sum_{k}(v_{i,k}-\bar{v_i})(v_{j,k}-\bar{v_j})}{\sqrt{\sum_{k}(v_{i,k}-\bar{v_i})^2}\sqrt{\sum_{k}(v_{j,k}-\bar{v_j})^2}} \quad (8)$$

where $v_{i,k}$ denotes the QoS value that user i received from service k, v_i denotes the average QoS that user i received from all the services invoked, and the summation is over all the services that have been invoked by both i and j.

In the real life scenarios we observe that it is rather rare to have a consumer that has a larger number of similar consumers or a service that has large number of similar services. Hence, we will be working with a very limited data set. The prediction confidence could be increased if we use both content and user based similarity to predict the observed QoS values. In our approach when we have a a missing QoS value for a service and we do not have any similar users that have invoked that particular service, we find out similar services and use their QoS values to predict the missing QoS value, and vice versa. We assume that $S_i \neq \emptyset$ and $U_p \neq \emptyset$. We use f_u and f_i to assign weightage to both the filtering values.

$$f_u = \sum_{i \in S_i} \frac{Sim(i,j)}{\sum\limits_{i \in S_i} Sim(i,j)} \times Sim(i,j) \tag{9}$$

We use $\lambda (0 \leq \lambda \leq 1)$ to adjust the influence of both user(u) based and content based(i) filtering.

$$\begin{aligned} W_u &= \frac{f_u \times \lambda}{(f_u \times \lambda) + (f_i \times (1-\lambda))} \quad and \\ W_i &= \frac{f_i \times \lambda}{(f_i \times \lambda) + (f_u \times (1-\lambda))} \end{aligned} \tag{10}$$

where $W_u + W_i = 1$, hence the predicted QoS value would be

$$\begin{aligned} Q_{val} = W_u \times (\bar{v}_i + \frac{\sum\limits_{j \in S_i} Sim(i,j)(v_{j,p} - \bar{v}_j)}{\sum\limits_{j \in S_i} Sim(i,j)}) + \\ W_i \times (\bar{v}_q + \frac{\sum\limits_{q \in U_p} Sim(p,q)(v_{p,q} - \bar{v}_q)}{\sum\limits_{j \in U_p} Sim(p,q)}) \end{aligned} \tag{11}$$

Using the above mentioned approach implies that the initial set of services that are being considered for selection process have a very high chance of forming a service agreement. The observed and predicted QoS values help eliminate the uncertainty in perceived QoS values. The next step is to formulate a globally optimal composition using the dependency modeling and the services with better predicted observed values.

5 MULTI-CONSTRAINT MODELING

The initial filtering process provides us with a good platform of a subset of services to be considered for service composition. For service composition we use the concept of Markov Decision Process (MDP). MDP is an Artificially Intelligent model for making decisions in environments where there is a higher percentage of uncertain outcomes. MDP has been successful applied in different domains to solve decision making problems [8] [9] [10] [11]. We model each dependency relationship as a single constraint and use this model to solve the global optimization dependency problem for QoS parameters of component Web services.

In our system each Web service has a unique identifier ID and a QoS vector. Each QoS vector is a combination of five QoS values i.e. QoS = <Cost, Execution-Time, Reliability, Availability, Throughput> (it can easily be generalized to more QoS component values since $QoS =< QoS_1, QoS_2, .., QoS_n >$). Markov Decision Process is defined as $< S, A, P, R, \gamma >$. S represents the set of states of the system; A stands for the set of actions in the system and $A(s)$ is a subset of action available at state s i.e. $A(s) \in A$ where $s \in S$; P is the probability of an action such that $P_a(s,s') = Probability(s_{t+1} = s' \mid s_t = s, a_t = a)$; R is the expected or immediate reward for the current transition and γ is used to factor in the importance of current reward and the future rewards. We can extend this basic single constraint MDP to a multi-constraint MDP by enhancing the reward function to consider the multiple constraints. We modify the reward function to compensate for multiple constraints. Let us assume that the system has c number of constraints. The reward function will be as follows:

$$R_a(s,s') = [R1_a(s,s'), R2_a(s,s'), ..., Rc_a(s,s')]^T \tag{12}$$

In addition to the above constraints we need to consider the fact that every composite system has an entry and exit point. We can argue that this entry and exit point may contain more than one service. Some systems can be invoked using multiple services and similarly their execution may take multiple routes to reach different end states. Hence, the updated model for Web service will have 7 tuples $MDP =< S, S_s, S_e, A, P, R, \gamma >$. Where S_s is a set of starting states for the composition and S_e is the set of end states of the composition where $S_s \in S$ and $S_e \in S$. Here an action corresponds to the execution of a Web service and the reward vector contains one reward for every QoS component of the Web service. For Web service ws and our QoS attributes under consideration the reward vector will be:

$$\begin{aligned} ws(q)(s,s') = [ws(q_{co})(s,s'), ws(q_{et})(s,s'), ws(q_{re})(s,s'), \\ ws(q_{av})(s,s'), ws(q_{th})(s,s')]^T \end{aligned} \tag{13}$$

The solution of MDP is a decision constraint and a constraint is the procedure for selecting Web services. These constraints, represented by ζ are basically just mappings from states to actions, defined as $\zeta : S \to A$. Each constraint can represent a single composite solution of Web service. Hence, our system searches for a set of Pareto optimal constraints which optimize QoS attributes of the composite system. The set ζ^o of Pareto optimal constraints is defined by:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589244, IEEE Transactions on Services Computing

$$\zeta = \left\{ \zeta^{O} \in \coprod \mid \nexists \zeta \in \coprod, s.t. V^{\zeta^{O}}(s) >_o V^{\zeta}(s), \forall s \in S \right\} \tag{14}$$

where $\coprod$ defines the set of all constraints and dominance relation is represented by $>_o$. For $a = (a_1, a_2, ..., a_n)$ and $b = (b_1, b_2, ..., b_n), a >_o b$ means that $a_i \geq b_i$ is satisfied for all i and $a_i > b_i$ for at least one i. Moreover $V^{\zeta}(s) = (V_1^{\zeta}(s), V_2^{\zeta}(s), ..., V_m^{\zeta}(s))$ is the value of the vector s as per the constraint ζ i.e.:

$$V^{\zeta}(s) = E_{\zeta} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \tag{15}$$

where E_{ζ} represents the expected value when the service follows constraint ζ, s_t at time t with the reward vector r_t.

Q-learning is calculated as:

$$Q_{\zeta}(s, a) \models E_{\zeta} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\} \tag{16}$$

5.1 Single constraint Multi-objective Service Composition

In our approach we use Q-learning for QoS objectives. We spawn a Q-learning service for every QoS objective. Based on the multiple objective criteria the importance (weight) of every QoS objective for a Web service is learned rather than using predefined weights. Every service i selects a Web service ws_i at each state S in such a way, that optimizes the relative QoS objective of the Web service. Once this process is completed then the Web service assign a weight $Wi(s)$ to their selected service and later negotiate among themselves to select the most suitable candidate to be executed at each state. The service having the maximum weight will be able to execute its option at each state. The objective is:

$$W_k(s) = Max_{i \in 1,....,n} W_i(s) \tag{17}$$

Therefore, in this case the Web service k is called the leader service and is allowed to invoke Web service $W_k(s)$. In the next round the Web services evaluate the results of the previous selections and adjust their $w_i(s)$ values based on positive or negative outcome of the previous round. Hence, we may have a different leader service in the next round.

W represents the difference in the predicted versus actual reward received by the Web service. Web services predict to receive a reward value P if their selected service was executed at s. If their service was not executed then instead of receiving the predicted reward P it receives the actual reward A. So W = P - A. In the case when a service's suggested selection was executed P = A and if not then the service will receive a negative reward that is equal to (P - A). So if service k ends up being the leader in a certain round then all service except k will update their W values using the following:

Algorithm 2 Single constraint Algorithm

```
Start at state s
randomly select a service k to be the leader
W_k ← 0
a_k ← argmax_a Q_k(s, a)
repeat
    for all services i except k do
        W_i ← max_a Q_k(s, a) − Q_i(s, a_k)
        if the highest W_i > W_k then
            W_k ← W_i
            a_k ← argmax_a Q_i(s, a)
            k ← i
        end if
    end for
until converges
return b
```

$$W_i(x) \rightarrow (Q_i(x, a_i) - (r_i + \gamma max_{b \in a} Q_i(y, b))) \tag{18}$$

At this point the next state "s'" and the reward vector r_i is influenced by the leader service rather than being a decision of each individual service. We present this in Algorithm 2.

5.2 Multiple constraint Multi-objective Service Composition

In our second approach we introduce the concept of convex hull to solve the multiple constraint problem for service selection for composite solutions. The convex hull is defined as the smallest convex set that contains all the points that lie on the boundary of this convex set. We combine this concept, which is similar to the Pareto front, into our Q-learning based approach based on the fact that both concepts are essentially maxima over different trade-off factors in the linear domains.

We use a value iteration based method to obtain a set of service selection constrains that is Pareto optimal:

$$V(s, a) = V(s, a)(1 - \alpha) + \alpha \left[\gamma \mathcal{R} \bigcup_{a'} V(s', a') + r(s, a) \right] \tag{19}$$

$V(s, a)$ represents the set of vertices of the $\mathcal{R}$ when action a is taken at state s for all possible Q-value vectors , r is the reward vector, discount value for the process is represented by γ, rate of learning is controlled by α and the operator $\mathcal{R}$ represents the set of extracted vertices of the $\mathcal{R}$.

Our solution follows the greedy exploration methodology and the dominance relationship between Q-vectors is used for selecting a particular action. In this approach we do not backtrack based on the maximal expected reward for each vector rather we use the set of maximal expected rewards for the given set of constraints as the basis for backtracking. This is illustrated in Algorithm 3.

Algorithm 3 Multiple constraint Algorithm

1: $V(a,s)$ = Randomly initialize $\forall s, a$
2: **while** convergence condition not met **do**
3: **for all** $s \in S, a \in A$ **do**
4: $V(s,a) = V(s,a)(1-\alpha) + \alpha\left[\gamma \mathcal{R} \bigcup_{a'} V(s',a') + r(s,a)\right]$
5: **end for**
6: **end while**

6 Study and Results

To study the efficiency of our approach we implemented our motivational scenario. We compared the single constraint and multi-constraint approaches along with our invocation patterns and collaborative filtering based solution. Moreover, we compared our approach with similar approaches in the literature. The experiment environment consists of a Windows server 2008 (SP2)-based Quad core machine with 8.0 GB of ram. We used the WSDream-QoS dataset [1], which contains multiple Web services distributed in computer nodes located all over the world (i.e., distributed in 22 different countries). Planet-Lab is employed for monitoring the Web services. We take the published services and their corresponding QoS values and assign cost values to them.

6.1 Experiment 1: Single Constraint Algorithm

In the first experiment, we compared the single constraint algorithm with the Q-learning approach [12]. We compare the effectiveness of our approach in formulating a composite solution with no pre-defined user weight preferences against the user defined weight vector for the Q-learning approach. We use the accumulated reward for a composition to compare the quality of the discovered solution. The Q-learning approach uses a weight vector of $\omega = (q_{co} = 0.2, q_{et} = 0.2, q_{re} = 0.2, q_{av} = 0.2, q_{th} = 0.2)$ for this experiment i.e equal weightage for all the QoS components. Fig. 3 shows the results of our experiment of average results of 30 runs of both the algorithm with varying number of Web services. We can see that the single constraint approach out performs the Q-learning approach regardless of the number of Web services. The difference in quality of solutions increases when the number of Web services increase. This is attributed to the fact that our approach scales better at exploring the Pareto front than the Q-learning approach. Secondly, pre-defined weights guide the search process for the Q-learning based approach with may not be the best case solution since learning the weights on the fly could find solutions in other wise unexplored regions.

6.2 Experiment 2: Multi-Constraint Algorithm

In this set of experiments, we demonstrate that our algorithm finds Pareto optimal solutions considering

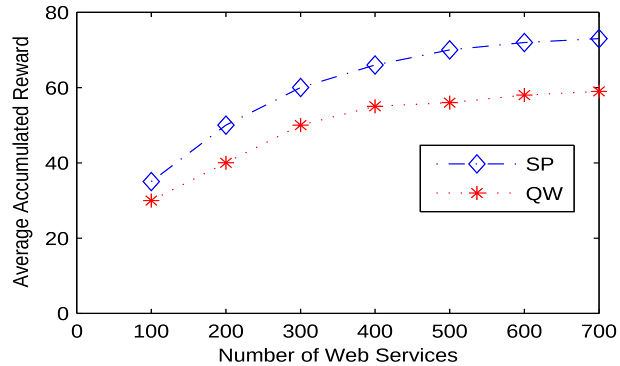

Fig. 3. Single Constraint Algorithm

the dependency relationships among the different QoS components. We assign multiple concrete services (50 and 100) respectively to the abstract services and observe the proposed solutions. In the first experiment, we assigned 50 concrete Web services to every abstract component Web service. We use three QoS attributes i.e. Cost, Availability and Response time. The objective is to minimize the cost and response time, while maximizing the availability of the system. Figure 4 shows the results of the pareto-optimal solutions found. We can see that the proposed algorithm is able to find high quality solutions.

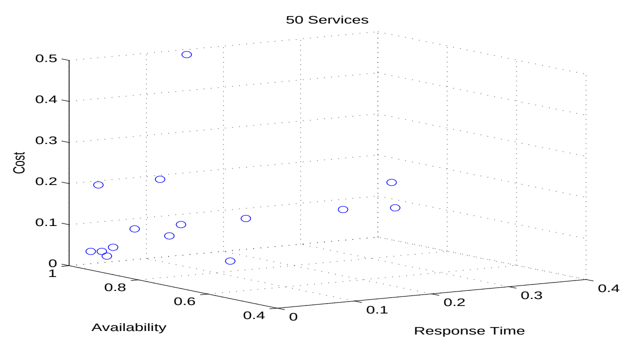

Fig. 4. Results of composition with 50 services

In the next experiment, we increased the number of concrete services from 50 to 100.

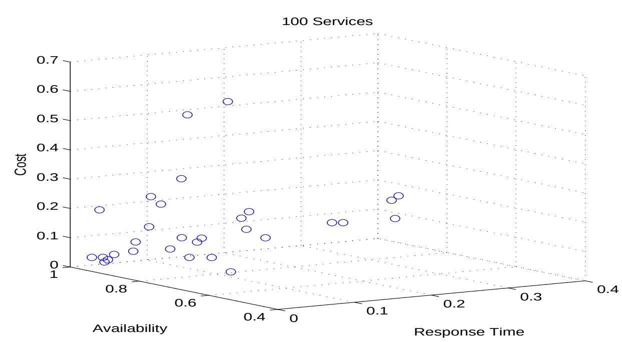

Fig. 5. Results of composition with 100 services

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589244, IEEE Transactions on Services Computing

In the next experiment, we show that our solution converges very effectively towards the solution with varying number of services. We tested our approach wi four abstract Web services and varied the number concrete services to 100, 200, 300 and 400 Web service As we can see from Fig. 6, our solution takes long time for finding optimal solutions when the number concrete Web services are increased. With 100 concre Web services, our solution converges at around 60 episodes mark, while when we increase the numb of concrete Web service to 200 it take it about 80 episodes to converge. Similarly, it takes 1200 and 140 episodes to find solutions for 300 and 400 concrete We services respectively. This shows that our approach ca handle a large number of Web services and still perforn efficiently.

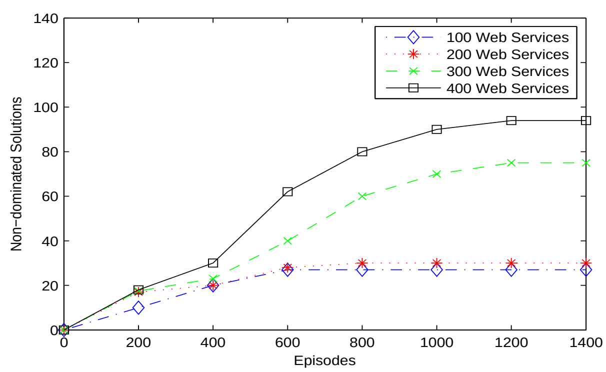

Fig. 6. Multiple Constraint Algorithm

6.3 Experiment 3: Individual QoS value comparisons

In this set of experiments, we show the difference in performance of solution when we use just single constraint MDP versus multiple constraint MDP and adding the information from dependency modeling and collaborative filtering. Figure 7 shows the individual QoS value comparisons for our proposed approach. We can see the performance of multiple constraints , multiple constrains with invocation patterns and multiple constrains with invocation patterns with filtering on the different QoS values of Cost, Execution Time, Reliability, Availability and Throughput when tested on a pool of 50 to 400 service. We can see that Multi-policy with invocation patterns with filtering consistently yields better results for all the QoS values. We can see that introducing invocation patterns into multiple constrains algorithm yields significant improvement for QoS value of Cost, Reliability and Availability. These QoS values rely heavily on the orchestration pattern of service within a composition. Reliability benefits most from the introduction of collaborative filtering as compared to other methods. Availability was the least improved QoS value among our experiments as the services did not show any significant variation in their availability values. However, it is evident that adding the invocation information and predicting observed QoS values improves the overall

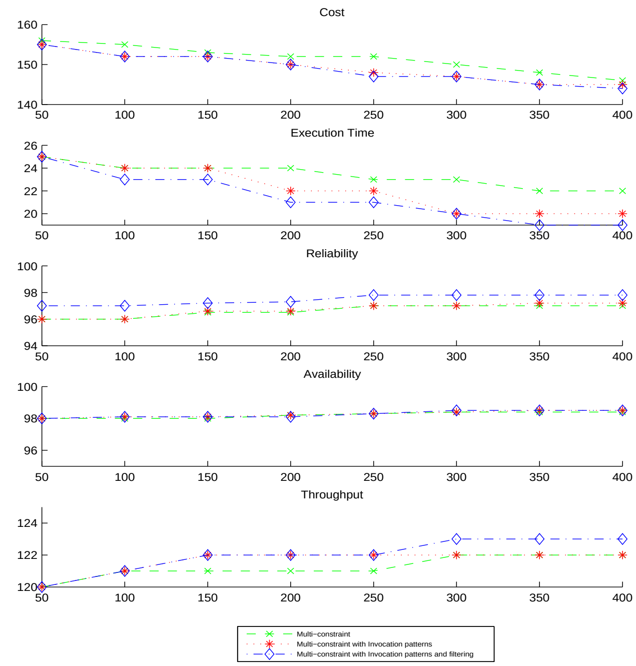

Fig. 7. QoS value comparison

6.4 Experiment 4: Individual QoS value comparisons for Invocation patterns

In this set of experiments we compare the impact of using our approaches for different QoS invocation patterns. We implemented three different SOA systems that use 15, 30 and 45 services respectively with different invocation patterns. The number of distinct execution paths for each of these systems is 16 , 51 and 77 respectively. We replace one service from the composite system at different invocation points to measure the impact on the total utility of the system. Figure 8 - Figure 12 show the utility gain for all three solutions with varying number of component services. We can see that for every invocation pattern MclpFi performs better when the number of component service increase. This gain is primarily related to better prediction since the more data points are available to calculate the similarity of services. Cost, Execution Time and Throughput are the components that benefit more from the proposed solution. Reliability and Availability also show slight gains in the utility value.

6.5 Experiment 5: Comparison with existing approaches

In this experiment we compare our solution(MclpFi) with similar approaches presented in the literature. We base our comparison on the utility values of these techniques and the time it takes to reach a solution.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589244, IEEE Transactions on Services Computing

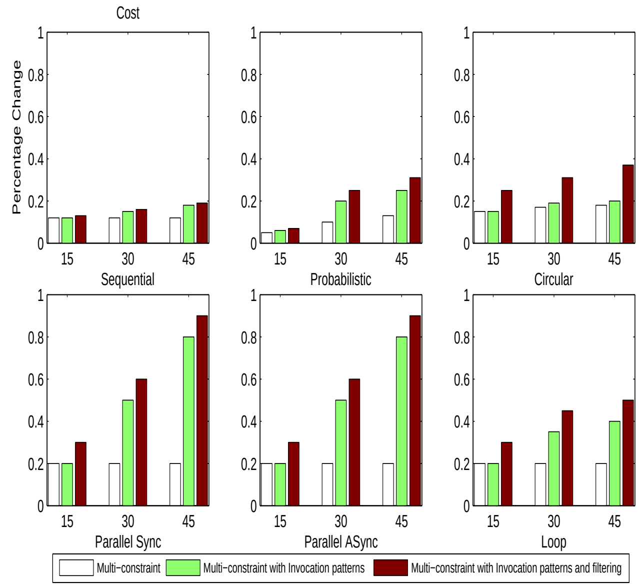

Fig. 8. Utility comparison of Cost

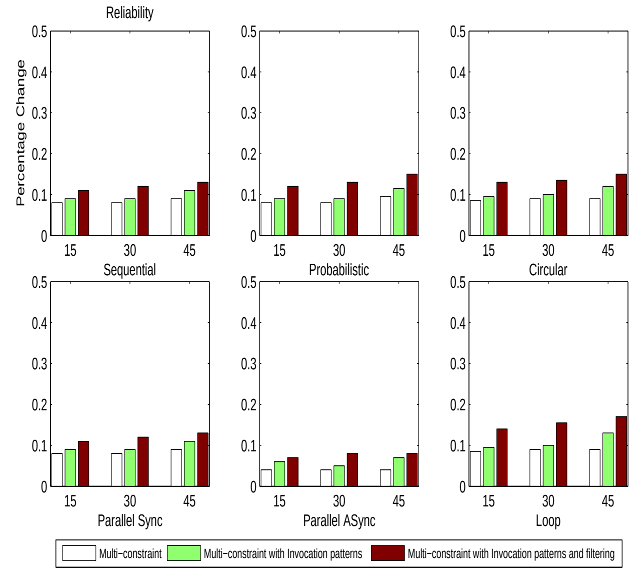

Fig. 10. Utility comparison of Reliability

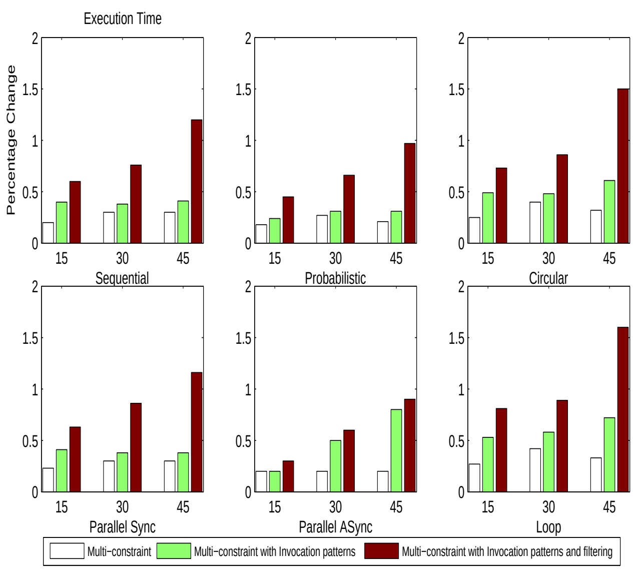

Fig. 9. Utility comparison of Execution Time

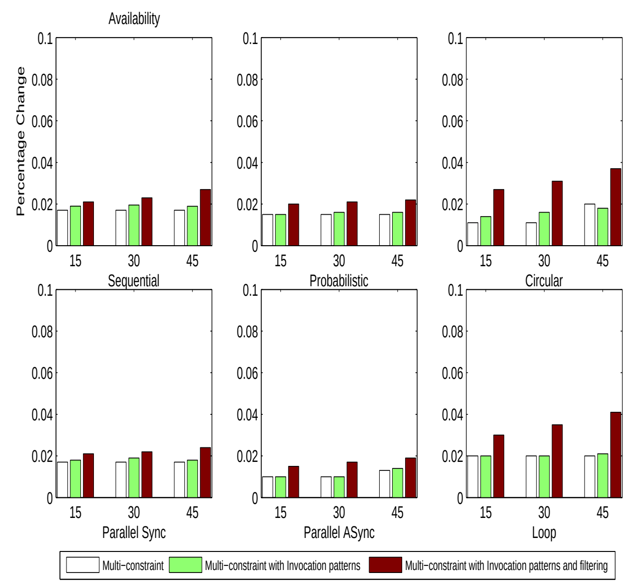

Fig. 11. Utility comparison of Availability

SBA [13] uses a GA based approach with an offer and counter-offer based protocol for searching a mutually agreeable solution. The degree of overlap among the QoS values requested by the consumer and those offered by the provider are taken into account in SBA. We use the results for the maximum overlap (80%) in our comparisons. Similarly, NBA [14] uses a GA based approach with a very similar fitness function as used in our technique, but does not take into consideration any other parameters. SWC [15] also uses a GA based approach for the semantic composition of Web services. It uses the semantic equivalence in addition to the QoS values to determine the best offering for the composition. BLGAN [16] uses a Bayesian learning based approach with GA and incomplete information model to learn the reserve price of it opponent. GTFSN [17] presents a game theoretical model of signaling games for Service Level Agreement negotiation. The results are presented in Figure 13. We can see that our solution is the quickest in improving on the initial solution. This can be attributed to the use of invocation patterns, and the solution improves exponentially as the values stabilize (high jumps around time 27 and 49). We can also see that our solution finds a solution within the 97% utility range in about 66ms, while SWC (the second best) takes about 3 times more time. Other techniques fail to generate such a solution and NBA and SBA plateau around the 92% range while BLGAN and GTFSN only reach a solution of around 94% utility. The results suggest that our approach outperforms similar methods both in terms of finding the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2589244, IEEE Transactions on Services Computing

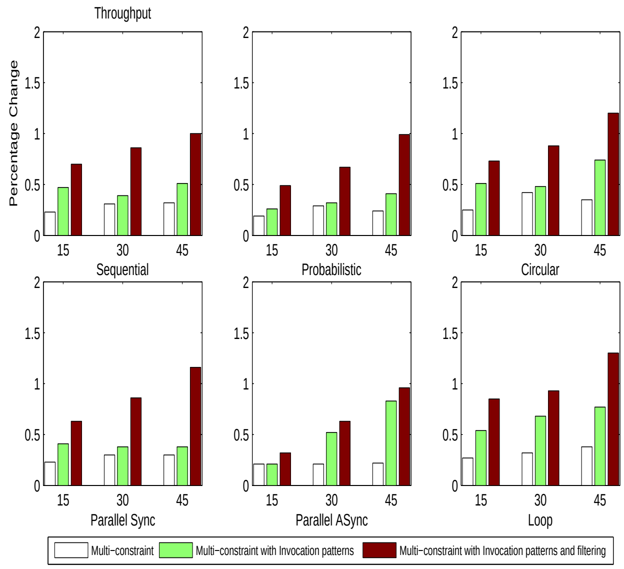

Fig. 12. Utility comparison of Throughput

optimal solution and the amount of time it takes to find that solution.

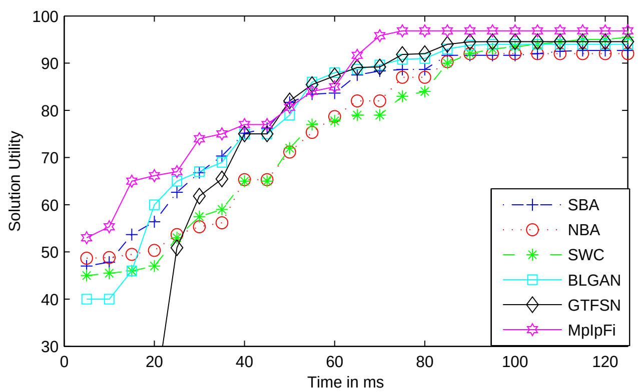

Fig. 13. Performance comparison of proposed approach

7 Literature Review

QoS based service selection has been an active research topic for service composition. Most of the research in this area has focused on local optimization of QoS attributes that may not result in a globally optimal solution [18] [19] [20]. Some early efforts of global optimization using integer programming, heuristic based searches and critical path have been presented in [21], [22], [23], [24].

A mixed integer programming based global optimization approach has been presented in [25]. In this approach, the authors decompose the global constraints into local constraints by mapping them to a set of pre-computed local QoS values. This approach provides locally efficient component services that are then combined to formulate the composition. A major shortcoming of this approach lies in the fact that all QoS dimensions are considered independently, and no dependency or correlations among these components is taken into consideration. Second, in some scenarios where the QoS requirements are very aggressive, the approach translates the global requirements into very constrained local requirements. Thus, the algorithm fails to finds a solution where as a solution adhering to the global constraints may exist in the system.

Canfora et al. [26] proposed a Genetic Algorithms (GAs) based QoS-aware composite service binding and rebinding approach. This approach allows the service orchestrators to apply non-linear QoS aggregation formulae as compared to linearization for the traditional approaches. However, their solution focuses on collecting usage pattern data to predict the need of re-binding for different invocation instances. This increases the system overhead and the service needs to be a part of the composition for it to be optimized. Secondly the proposed solution assumes that there will always be a solution that will meet all the requirement. Yu et al. [27] present a broker based solution for QoS based service selection. They present user-defined utility function for both of their models i.e. combinatorial and graph based model. They consider invocation patterns in their utility functions and present different heuristic based approaches to find near optimal solutions. However, they present two different solutions for sequential executions and complex structures i.e. loop etc and rely solely on the published QoS values. Zeng et al. [28] present an Integer Programming based solution that finds optimal service composition solutions. However, in case of multi-path scenarios they only optimize the solution based on the path with the highest probability as well as their definition of critical path uses the worst case scenarios of all the service in the execution path, which may not be a good approximation and hence is not suitable for large systems or systems with dynamic service needs.

In [29] authors propose a discrete probability distribution based model to solve the uncertainty of QoS values of the services. They incorporate composition control along with probability distribution on a uniform framework. The proposed approach works better than either using the mean or median and is independent of the any normality or uniformity constraints. This allows to directly convert the observed behavior in inputs for the system and a single analysis could approximates the results that could have been obtained by exhaustive black box simulations of the composition. This method however does not consider the dependency modeling behavior of the system and assumes that the probability distribution is independent of the factors like location of users and any other differentiating factors. In [30] present a soft constraint based solution for QoS service selection. They model selection characteristics using soft constraints, assign preferences to the penalties and constraints. Then the cumulative ranking of the above mentioned factors to compute the composite rank of the solution. This allows the users more flexibility incase services do not meet the exact composition criteria.

However, the proposed approach does not consider the effect of dependency modeling and leaves it to the user to use a simple ranking based solution.

Collaborative Filtering (CF) approaches i.e. both memory based and model based algorithms have been widely used in recommendation systems [31], [32], [33], [34]. Memory based approaches use the historic values of a user to recommend similar items [35], [36] on the other hand model based approaches [37],[38], [39], [40] apply different machine learning and statistical methods to learn user rating behaviors. Memory based approaches are relatively easier to implement, do not need any training set and can easily accommodate new rating data from users. However, they do not scale well as the data set grows. On the other hand item based approaches usually outperform memory based approaches in terms of scalability and quality of recommendation but they suffer from the fact that the model needs to be updated/ rebuild when new data points are received. There have been efforts to use collaborative filtering in service composition. Margaris et al. [41] use collaborative filtering for QoS requirements for WS-BPEL scenarios. They combine the collaborative filtering based score and the QoS requirements of the users to calculate an aggregate score for WS-BEPL scenarios. Karta [42] have used both Pearson correlation and vector based similarity approaches in collaborative filtering on MovieLens data set for service selection, comparing their work with multidimensional recommender system. However, this approach uses static QoS ontology and ranking registry data.

8 Conclusion

In this paper we presented a multi-objective QoS optimization approach that uses the structure of a service based system by incorporating different invocation patterns, and uses collaborative filtering to predict the QoS values of different services to formulate an optimized solution. Using the orchestration pattern of a composition allows valuable information that could be exploited to further optimize different QoS value such as Throughout and Execution time. Our proposed approach uses this information and searches for services that are best suited for the current orchestration, hence providing a very optimal solution for the composition at hand. Since this approach is highly reliant on the observed QoS values of a Web service, we employed collaborative filtering to fill in the gaps and predict the QoS values for the missing service interactions. The experiment results show the advantage of our approach when compared to other methodologies.

Acknowledgment

This work was supported by Grant xxxxxxx

References

[1] Y. Zhang, Z. Zheng, and M. R. Lyu, "Wsexpress: A qos-aware search engine for web services," in *Proc. IEEE Int'l Conf. Web Services (ICWS'10)*, pp. 83–90, 2010.

[2] Z. Malik and A. Bouguettaya, "Rateweb: Reputation assessment for trust establishment among web services," *The VLDB Journal*, vol. 18, no. 4, pp. 885–911, 2009.

[3] M. Papazoglou and P. Hall, *Web Services: Principles and Technology*, vol. ISBN: 978-0-321-15555-9. 2008.

[4] D. A. D'Mello and V. S. Ananthanarayana, "A tree structure for web service compositions," in *COMPUTE '09: Proceedings of the 2nd Bangalore Annual Compute Conference*, (New York, NY, USA), pp. 1–4, ACM, 2009.

[5] D. A. Menasce, "Composing web services: A qos view," *IEEE Internet Computing*, vol. 8, pp. 88–90, 2004.

[6] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 793–818, 2009.

[7] M. R. McLaughlin and J. L. Herlocker, "A collaborative filtering algorithm and evaluation metric that accurately model the user experience," in *SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Sheffield, UK, July 25-29, 2004*, pp. 329–336, 2004.

[8] N. Mastronarde, K. Kanoun, D. Atienza, P. Frossard, and M. van der Schaar, "Markov decision process based energy-efficient on-line scheduling for slice-parallel video decoders on multicore systems," *Multimedia, IEEE Transactions on*, vol. 15, no. 2, pp. 268–278, 2013.

[9] H. E. Carter, A. Martin, D. Schofield, G. Duchesne, A. Haworth, C. Hornby, M. Sidhom, and M. Jackson, "A decision model to estimate the cost-effectiveness of intensity modulated radiation therapy (imrt) compared to three dimensional conformal radiation therapy (3dcrt) in patients receiving radiotherapy to the prostate bed," *Radiotherapy and Oncology*, vol. 112, no. 2, pp. 187–193, 2014.

[10] M. Pesce, D. Munaretto, and M. Zorzi, "A markov decision model for source video rate allocation and scheduling policies in mobile networks," in *Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual Mediterranean*, pp. 119–125, IEEE, 2014.

[11] J. Patrick, "A markov decision model for determining optimal outpatient scheduling," *Health care management science*, vol. 15, no. 2, pp. 91–102, 2012.

[12] H. Wang, X. Zhou, X. Zhou, W. Liu, W. Li, and A. Bouguettaya, "Adaptive service composition based on reinforcement learning," in *Service-Oriented Computing*, pp. 92–107, Springer, 2010.

[13] E. D. Nitto, M. D. Penta, A. G., G. Ripa, and M. L. Villani, "Negotiation of service level agreements: an architecture and a search-based approach," 2007.

[14] X. Niu and S. Wang, "Genetic algorithm for automatic negotiation based on agent," in *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pp. 3834 –3838, june 2008.

[15] F. Lecue, U. Wajid, and N. Mehandjiev, "Negotiating robustness in semanticweb service composition," in *Seventh IEEE European Conference on Web Services*, 2009.

[16] M. K. Sim, Y. Guo, and B. Shi, "Blgan: Bayesian learning and genetic algorithm for supporting negotiation with incomplete information," *Trans. Sys. Man Cyber. Part B*, vol. 39, pp. 198–211, Feb. 2009.

[17] C. Figueroa, N. Figueroa, A. Jofre, A. Sahai, Y. Chen, and S. Iyer, "A Game Theoretic Framework for SLA Negotiation,"

[18] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web services architecture," 2004.

[19] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: an introduction to soap, wsdl, and uddi," *Internet Computing, IEEE*, vol. 6, no. 2, pp. 86–93, 2002.

[20] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web services description language (wsdl) version 2.0 part 1: Core language," *W3C Recommendation*, vol. 26, 2007.

[21] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, "Constraint driven web service composition in METEOR-S," in *Services Computing, 2004.(SCC 2004). Proceedings. 2004 IEEE International Conference on*, pp. 23–30, IEEE, 2004.

[22] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for QoS-aware web service composition," in *Web*

Services, 2006. ICWS'06. International Conference on, pp. 72–82, IEEE, 2006.

[23] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *Software Engineering, IEEE Transactions on*, vol. 33, no. 6, pp. 369–384, 2007.

[24] L. Ai, M. Tang, and C. Fidge, "Partitioning composite web services for decentralized execution using a genetic algorithm," *Future Generation Computer Systems*, vol. 27, no. 2, pp. 157–172, 2011.

[25] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *Proceedings of the 18th international conference on World wide web*, pp. 881–890, ACM, 2009.

[26] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "A framework for qos-aware binding and re-binding of composite web services," *Journal of Systems and Software*, vol. 81, no. 10, pp. 1754–1769, 2008.

[27] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, p. 6, 2007.

[28] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, pp. 311–327, May 2004.

[29] D. Ivanovic, P. Kaowichakorn, and M. Carro, "Towards QoS prediction based on composition structure analysis and probabilistic environment models," in *Principles of Engineering Service-Oriented Systems (PESOS), 2013 ICSE Workshop on*, pp. 11–20, IEEE, 2013.

[30] M. A. Zemni, S. Benbernou, and M. Carro, "A soft constraint-based approach to qos-aware service selection," in *Service-Oriented Computing*, pp. 596–602, Springer, 2010.

[31] R. Burke, "Hybrid recommender systems: Survey and experiments," *User modeling and user-adapted interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[32] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003.

[33] H. Ma, I. King, and M. R. Lyu, "Effective missing data prediction for collaborative filtering," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 39–46, ACM, 2007.

[34] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175–186, ACM, 1994.

[35] A. Krzywicki, W. Wobcke, Y. Kim, X. Cai, M. Bain, A. Mahidadia, and P. Compton, "Collaborative filtering for people-to-people recommendation in online dating: Data analysis and user trial," *International Journal of Human-Computer Studies*, vol. 76, no. 0, pp. 50 – 66, 2015.

[36] R. Jin, J. Y. Chai, and L. Si, "An automatic weighting scheme for collaborative filtering," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 337–344, ACM, 2004.

[37] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen, "Scalable collaborative filtering using cluster-based smoothing," in *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 114–121, ACM, 2005.

[38] T. Hofmann, "Collaborative filtering via gaussian probabilistic latent semantic analysis," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pp. 259–266, ACM, 2003.

[39] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 89–115, 2004.

[40] T. Hofmann and D. Hartmann, "Collaborative filtering with privacy via factor analysis," in *Proceedings of the 2005 ACM Symposium on Applied Computing*, pp. 791–795, 2005.

[41] D. Margaris, C. Vassilakis, and P. Georgiadis, "An integrated framework for adapting ws-bpel scenario execution using qos and collaborative filtering techniques," *Science of Computer Programming*, vol. 98, Part 4, no. 0, pp. 707 – 734, 2015.

[42] K. Karta, "An investigation on personalized collaborative filtering for web service selection," *Honours Programme thesis, University of Western Australia, Brisbane*, 2005.

Khayyam Hashmi is a Ph.D. candidate in the Department of Computer Science at Wayne State University. He is a member of the Services COmputing REsearch (SCORE) Laboratory. His research interests include service computing, distributed systems, semantic web and software process engineering.

Zaki Malik is an Assistant Professor of computer science at Wayne State University, where he heads the Services COmputing REsearch (SCORE) Laboratory. His research interests lie broadly in service computing, reliable distributed systems, Web databases, and semantic integration systems.

Abdelmounaam Rezgui is an Assistant Professor in the Department of Computer Science at New Mexico Tech University. His research interest include High Performance Spatial Cloud Computing, Spatio-Temporal Data Intensive Applications on the Clouds, Data and Service Quality, Power Saving Techniques for Cellular Networks, Service-Oriented Sensor Networks, Semantics-Based Data Interoperability and Integration.

Abdelkarim Erradi is an Assistant Professor in the Department of Computer Science and Engineering at Qatar University. His research interest include Cloud Computing particularly,Software Composition for Distributed Systems ,autonomic computing systems, Policy-based management of Middleware systems particularly Web Services and Web service technologies .