Efficient QoS Aggregation in Service Value Networks

Steffen Haak  
Research Center for Information Technology (FZI)  
haak@fzi.de

Benjamin Blau  
SAP AG  
benjamin.blau@sap.com

Abstract

In recent years, the trend towards standardization, simplification and modularization in the service sector has fostered the rise of Service Value Networks where providers and consumers jointly co-create value. With many different competing services available, the user experience, which is captured by the non-functional Quality-of-Service (QoS) attributes, is an important competitive factor. QoS computation for complex Web services, i.e., the aggregation of QoS factors from atomic services, is essential for an automated an optimized Web service selection process. However, the computational complexity of QoS aggregation has often been disregarded in the respective field of research, whereas computational efficiency is inevitable for the application of optimization approaches in on-line scenarios. The threefold contribution of this paper consists of an elaboration on the computational complexity of aggregating QoS, an approximation scheme that allows for a computational efficient optimization and a broad analytical and simulation-based evaluation of this approach.

1. Introduction

For a long time, the development and consumption of Web services have been complex activities exclusively performed by technical experts. With the raise of light-weight technologies such as RESTful architectures [1, 2] and “fat free” message exchange formats such as JSON [3], a trend towards radical simplification of service creation and consumption is observable. This trend fosters ecosystems of “prosumers” that collaboratively develop and consume dynamic services in networked environments, i.e., Service Value Networks (SVNs) [4]. SVNs are networked economies constituted of distributed service providers and consumers. In these environments, services are dynamically composed into value-added complex services fulfilling the need of a long-tail of individual customers [5].

One of the key characteristics of services in general is their intangible nature. More precisely, value creation through services is dominated by intangible elements [6]. The consumer of a service experiences the performance or activity which embodies the main portion of created value [7, 8]. From a technical perspective, such intangible elements experienced by the customer are embodied by the Quality-of-Service (QoS) dimension.

When dynamically composing distributed services into complex services, the computation of the QoS is a central task in SVN. Especially the computational complexity of this task has often been ignored in the respective field of research, whereas computational efficiency is inevitable for the application in on-line scenarios.

Such an on-line scenario could be a real-time service integration platform, allowing for an autonomic composition of complex services out of atomic services based on customer requests. Typically the customer specifies his functional needs by asking for a set of so-called candidate pools, which cluster functionally equivalent service alternatives. An example request could be for a complex service consisting of storage, billing and payment. The challenge is not only to find technically feasible solutions, moreover finding the optimal composition from a QoS perspective is a cumbersome task, requiring an efficient aggregation of QoS values from the single services.

Tackling the challenge of aggregating QoS in the context of service composition in SVN, the contribution of the work at hand is threefold: Laying the groundwork and identifying the current research gap, we (i) elaborate the computational complexity of aggregating QoS depending on feasible attribute types and assess possible optimization approaches. We (ii) provide an optimization approach based on an approximation scheme that restores computational efficiency in case of attribute types with NP-hard complexity regarding their aggregation. Finally, we (iii) evaluate our approach analytically and simulation-based regarding the error that arises from the approximation in different deterministic and stochastic scenarios.

The remainder of this work is structured as follows. Section 2 covers the related work in this area. We then present in Section 3 the foundations and challenges of aggregating QoS by describing typical classes of QoS attributes and common approaches for optimizing...
QoS-aware service compositions by means of Graph algorithms and linear programming techniques. Section 4 contains our main contribution, a computational efficient approximation for aggregating multiplicative attributes along with a broad evaluation of the approximation following in Section 5. The paper is concluded in Section 6 with remarks on the practical implications and an outlook on our next steps within this research.

2. Related Work

The challenge of aggregating QoS attributes of atomic services in various process patterns is a well-investigated field in BPM and service composition research.

In the context of automatic service composition different approaches have been proposed in recent literature. Mostly backward chaining is applied to derive suitable compositions starting from a central objective [9, 10]. In most of the cases such models are based on formal description languages focusing on functional service characteristics as proposed by the W3C recommendation SAWSDL, OWL-S and WSMO. Other approaches such as [11] also incorporating the service lifecycle aspect and time dependencies. The main scope of all of these approaches is dealing with service functionality as the only criteria for composition and largely ignores other non-functional or QoS properties.

Focusing on mathematical models for aggregating quality attributes of single services into complex services depending on multiple types of process patterns are investigated in [12-15]. Although this stream of research considers different types of attributes and the implications for a corresponding aggregation algorithm, computational complexity and desired efficiency is not in scope of their investigation.

Another stream of research targets a more comprehensive solution for managing functional and non-functional (i.e. QoS) service characteristics across the entire lifecycle based on a model of atomic and composite services [16, 17]. Although outlined approaches also focus on automation and on-line computation of QoS aggregation, complexity aspects and efficient algorithm design is not in the focus.

[18] propose an linear programming (LP) approach that enables an automated Web service composition while maximizing the user experience which is modeled as QoS dependent utility function. The authors also describe how different types of QoS values can be aggregated in such an optimization scenario. They argue that probability values like the reliability or availability of a service can be aggregated using the following function:

$$q^* = \prod_{i=1}^{N} e^{q_i}$$

In order to include the aggregation into a LP formulation, the logarithm function is applied. However, this seems not to be reasonable, as the result is an addition of the QoS attributes, rather than a multiplication.

3. Foundation & Challenges

We identify five different types of attribute types that are categorized by their aggregation function (cf. Table 1).

<table>
<thead>
<tr>
<th>Type</th>
<th>Aggregation Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additive</td>
<td>$q^* = \sum_{i=1}^{N} q_i$</td>
</tr>
<tr>
<td>Multiplicative</td>
<td>$q^* = \prod_{i=1}^{N} q_i$</td>
</tr>
<tr>
<td>Average</td>
<td>$q^* = \frac{1}{N} \sum_{i=1}^{N} q_i$</td>
</tr>
<tr>
<td>Min</td>
<td>$q^* = \min(q_1, q_2, ..., q_N)$</td>
</tr>
<tr>
<td>Max</td>
<td>$q^* = \max(q_1, q_2, ..., q_N)$</td>
</tr>
</tbody>
</table>

Common representatives for the additive attributes are the response time or the costs of a service. Typical attributes that are aggregated in a multiplicative way are the availability or reliability of a service, which are expressible as probability values. Reputation or user ranks can be aggregated using the average function. Representatives for the min or max function are the throughput of a service or the encryption level. In the winner determination problem, as it occurs in multi-attributive auctions [19] or in the complex service auction [20], the welfare or utility maximizing configuration or path has to be calculated. In optimization scenarios for QoS-aware composition of Web services [18] or custom cloud services [21], the challenge is to find the optimal complex service configurations with respect to user-defined QoS preferences.

There are two different approaches to tackle this problem. In a more graph oriented approach, as it is suitable for finding the best offer in a Service Value Network, graph algorithms can be used to find the best offer by calculating the optimal path through the
network. [20] propose the use of the Dijkstra algorithm [22] to compute the shortest path within polynomial time complexity, however, this can only be achieved under the assumption of additive and monotone aggregation operations, as Dijkstra relies on the Bellman property [23]. This however hinders the inclusion of any QoS attribute other than the ones that fall into the class of additive aggregation functions (cf. Table 1). Algorithms, that calculate the optimal path in networks without the Bellman property, are known to be NP-hard. In order to capture other non-Bellman attribute classes, an extension to Dijkstra is proposed in [24], yet the NP-hard complexity is thereby traded for exponential space complexity. [18] and [21] user linear programming and linear integer programming techniques to compute the optimal service composition. Depending on the problem instantiation, these problems can still be computed in polynomial time. The objective function, however, is required to be of linear form. For maximizing or minimizing an additive QoS attribute, the objective function typically has the following form, where \( x_1, \ldots, x_n \) denote binary decision variables.

\[
\max \sum_{i=1}^{N} x_i \cdot q_i
\]

While there are linearization methods for both min and max function available, multiplicative attributes cannot be included into a linear objective function of such an optimization problem.

4. Proposed Solution

In order to cope with the challenges we presented in the preceding section, a linear function approximating the multiplication result is desirable. We thereby can make use of some neat properties of the attributes as they occur in our domain of interest.

QoS attributes that require the multiplication operation are commonly probabilities or ratios like the average availability, failure rate, etc. Common to these values is there domain which lies in the interval \([0, 1]\) and their proximity to the value of one. This is especially the case w.r.t. the availability of services or the probability of success of service components, which is the inverse of their failure rate. In almost all cases these values are above 99%.

Nonetheless, these values have to be aggregated, as there is a huge difference between an aggregated value of 99.5% and 99.9%. Seen as availability over the period of one year, this slight change would account for an additional outage time of 35 hours. In Web service scenarios where downtime is extreme costly, e.g. a banking service, these additional hours can quickly cost up to millions, thus a credible aggregated value becomes inevitable.

When composing a service out of several services or service components, we therefore need to aggregate all the individual QoS attributes to allow us to make a statement on the resulting overall performance of the complex service composition. The exact aggregation value \( q^* \) for \( N \) probability values \( q_1, \ldots, q_N \) (assuming stochastic independence of the attribute values of the different service components) can be calculated as follows:

\[
q^* = \prod_{i=1}^{N} q_i
\]

This aggregation function, being non-linear, however is not efficient computable in optimization scenarios with many different service composition alternatives. In those scenarios linear programming (LP) techniques or graph algorithms are applied. However they require linearity of the objective function for the LP approach or Bellman optimality respectively. In either case, solutions exist to linearize the problem or to cope with the state dependency within graph algorithms like Dijkstra [22], yet these solutions come at the cost of space complexity, thus not reducing the overall complexity, which is known to be NP-hard.

4.1. Approximation

For achieving an efficient computation, i.e. a solution lying within polynomial time complexity, a linear approximation can be of great benefit. Recapturing the typical characteristics of the attributes that are aggregated by multiplication, the following linear approximation function offers computational efficiency while simultaneously only yielding a small error, depending on the concrete problem instantiation:

\[
q^a = 1 - \frac{N}{\sum_{i=1}^{N} \alpha \cdot (1 - q_i)}
\]

\( q^a \) denotes the aggregated value, \( N \) the number of attributes, \( q_i \) are the attribute values and parameter \( \alpha \) can be used to parameterize the function according to the domain of attribute values, which allows to reduce the expected error of the approximation function. Values for \( \alpha \) that achieve a minimum expected error typically lie within the interval \([0, 1]\).

The function can be further simplified by pulling out the constants from the summation term:

\[
q^a = 1 - \alpha \cdot N + \alpha \cdot \sum_{i=1}^{N} q_i
\]

Other than the exact aggregation \( q^* \), this approximation is linear and thus can be included in any
5. Evaluation

We evaluate the proposed approximation analytically and by running different simulations. For measuring the error the following error function is defined:

\[ e(q^*, q^a) = \frac{|q^* - q^a|}{q^a} \]

Based on this error function, we conducted various sensitivity analyses with different parameter settings. For reducing the mathematical complexity and a better understanding of the reader, we set \( \alpha \) to a value of one for some parts of the evaluation, as we can draw the same conclusions without losing generality. We then obtain a simplified aggregation function with \( \alpha \) equaling one:

\[ q^a = 1 - N + \sum_{i=1}^{N} q_i \]

5.1. Deterministic Values

For a first analytic error estimation we assume the attribute values to be deterministic and parameter \( \alpha \) to be one. Thus the error can be calculated straightforward without the use of stochastic theory. In case all attribute values are equal, i.e. \( q_1 = q_2 = \ldots = q_N = q \), we obtain:

\[ e_1(q^*, q^a) = \frac{\prod_{i=1}^{N} q - 1 + N - \sum_{i=1}^{N} q}{\prod_{i=1}^{N} q} \]

This equation can be simplified to:

\[ e_1(q^*, q^a) = \frac{|q^N - 1 + N - N \cdot q|}{q^N} \]

The resulting error as a function of \( N \) and \( q \) is depicted in Figure 1. One can see that the error increases with \( N \) and decreases with \( q \).

However, in reality, not all values are equal. One way to measure this effect in a deterministic manner, is by spreading the \( N \) attribute values equally in the interval \([q - \varepsilon, q + \varepsilon]\), with

\[ q_1 = q - \varepsilon \\
\ldots \\
q_N = q + \varepsilon \]

We can derive the following error function, which shows the influence of the values being spread apart by two times \( \varepsilon \).

\[ e_2(q', q') = \prod_{i=1}^{N} q + \varepsilon \left( \frac{2 \cdot i - 2}{N-1} \right) - 1 + N - N \cdot q - \varepsilon \sum_{i=1}^{N} \left( \frac{2 \cdot i - 2}{N-1} \right) \]

Figure 2 shows that the error of the approximation actually first decreases with epsilon, until a value for \( \varepsilon \) is reached, where the approximation evaluates to the exact same result as the exact value of the product. After this point, the error increases again. The exact location of this root depends on the number of aggregated attributes \( N \) and their mean value \( q \).

The location of the root forms a valley where a decreasing value of \( q \) corresponds to an increasing value of \( \varepsilon \) (Figure 3). From a practical point of view, this is an advantageous property, as realistic values commonly “fall” into this valley.
Figure 2: Deterministic error in %, q=0.995

So far, we analyzed the error from an absolute point of view. In some scenarios, e.g. multi-criteria optimization problems, a utility or scoring function that transforms the aggregated value influences the consequent error in the application. A typical scoring function defines upper (b_U) and lower bounds (b_L) for the aggregated attribute value, with a linear utility progression in between these bounds:

\[
S(q^*) = \begin{cases} 
0 & q^* < b_L \\
\frac{b_U - q^*}{b_U - b_L} & b_L \leq q^* \leq b_U \\
1 & q^* > b_U
\end{cases}
\]

This transformation in fact zooms into the area of interest, scaling up small differences in the attribute value, thus also raising the expected error. Using the same simplification as above, we can derive the error of the approximation, which is only given for the range \(b_L \leq q^* \leq b_U\):

\[
e_3(S(q^*), S(q^a)) = \frac{1 - b_L - N - N \cdot q^*}{b_U - b_L} \frac{q^a - b_L}{b_U - b_L}
\]

By simplifying the above equation to

\[
e_3(S(q^*), S(q^a)) = \frac{1 - N + N \cdot q^* - q^N}{q^N - b_L}
\]

one can see that the error is actually independent on the upper bound \(b_U\), yet requires a limit analysis where \(q^*\) is close to \(b_L\). Figure 4 shows the error in dependency on \(b_L\) and \(q\). In most cases, it is not much higher than the error without using a scoring function, yet with a close proximity of \(q^*\) to \(b_L\), the error raises towards infinity. This implies that the proposed approximation function is not suitable, where the exact value of the aggregation function is very close to the lower bound of the scoring function. However, from a practical point of view, this case can be considered very unlikely.

5.2. Stochastic Values

In the last subsection we presented evaluation results that were relying on deterministic values for the QoS attributes that have to be aggregated. In reality, however, we are confronted with values that origin from some stochastic or random process with an underlying probability distribution. I.e. not all \(q_i\) are equal, nor are the spread evenly on a given interval. However, these deterministic considerations allowed for some much simpler analytic conclusions, which become far more complex in the stochastic case.

As the more sophisticated analytic considerations require a numerical integral evaluation with an increasing complexity in the number of variables, we also rely on Monte Carlo simulations within this subsection.

In the center of interest is again the error arising from using the approximation instead of the exact multiplication function. In the stochastic case, this error is not a deterministic value but a probability function. The expected value of this function thereby resembles a comparable result to the deterministic error value.
Figure 3: Deterministic error in % (N=5)

Figure 4: Error with scoring function in % (N=10)
This expected error, which is based on N independently distributed quality values, can be calculated by building the integral over all random variables. We thereby have to multiply the density functions of all random variables (i.e. the actual probability of obtaining a given value) times the error that results from this random variable instantiation. With \( f(q_i) \) being the probability density function of variable \( q_i \), we obtain the following formula:

\[
E\left(e\left(q^*, q^a\right)\right) = \int \cdots \int f_1(q_1) \cdots f_N(q_N) \cdot \frac{|q^*-q^a|}{q^*} \, dq_1 \cdots dq_N
\]

For two variables (N=2) which are distributed uniformly, i.e. \( q_1 \sim q_2 \sim U(a,b) \), we obtain:

\[
E\left(e\left(q^*, q^a\right)\right) = \int_a^b \int_a^b \left(\frac{1}{b-a}\right)^2 \frac{|1-2+q_1+q_2-q_1 \cdot q_2|}{q_1 \cdot q_2} \, dq_1 \, dq_2
\]

Figure 5 shows a numerical evaluation of the interval in dependence on \( a \) and \( b \). The worst case expected error in this figure with \( a=0.8 \) and \( b=0.9 \) evaluates to 3.162\%, whereas a typical setting with a distribution between \( a=0.99 \) and \( b=0.999 \) evaluates to a low expected error value of 0.003\%. Analogously one can estimate the expected error for other distributions and different numbers of variables. However the computational complexity for evaluating the integral strongly increases with \( N \).

Table 2 gives an overview on the expected error for the uniform distribution, for two to five variables and different parameters for the uniform distribution \((a, b)\).

<table>
<thead>
<tr>
<th>( N )</th>
<th>( a )</th>
<th>( b )</th>
<th>Error in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.99</td>
<td>0.348</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>0.999</td>
<td>0.003</td>
</tr>
<tr>
<td>3</td>
<td>0.9</td>
<td>0.99</td>
<td>1.085</td>
</tr>
<tr>
<td>3</td>
<td>0.99</td>
<td>0.999</td>
<td>0.009</td>
</tr>
<tr>
<td>4</td>
<td>0.9</td>
<td>0.99</td>
<td>2.257</td>
</tr>
<tr>
<td>4</td>
<td>0.99</td>
<td>0.999</td>
<td>0.019</td>
</tr>
<tr>
<td>5</td>
<td>0.9</td>
<td>0.99</td>
<td>3.911</td>
</tr>
<tr>
<td>5</td>
<td>0.99</td>
<td>0.999</td>
<td>0.031</td>
</tr>
</tbody>
</table>

In Table 3 the expected error for two and three normally distributed variables is shown, with different parameters for the mean value and variance of the normal distribution. All error values are calculated by means of numerical integration.

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \mu )</th>
<th>( \sigma^2 )</th>
<th>Error in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.1</td>
<td>1.979</td>
</tr>
<tr>
<td>2</td>
<td>0.9</td>
<td>0.01</td>
<td>1.238</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>0.1</td>
<td>0.698</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>0.01</td>
<td>0.014</td>
</tr>
<tr>
<td>2</td>
<td>0.999</td>
<td>0.1</td>
<td>0.666</td>
</tr>
<tr>
<td>2</td>
<td>0.999</td>
<td>0.01</td>
<td>0.006</td>
</tr>
<tr>
<td>3</td>
<td>0.9</td>
<td>0.1</td>
<td>5.560</td>
</tr>
<tr>
<td>3</td>
<td>0.9</td>
<td>0.01</td>
<td>3.988</td>
</tr>
<tr>
<td>3</td>
<td>0.99</td>
<td>0.1</td>
<td>1.296</td>
</tr>
<tr>
<td>3</td>
<td>0.99</td>
<td>0.05</td>
<td>0.317</td>
</tr>
<tr>
<td>3</td>
<td>0.999</td>
<td>0.1</td>
<td>1.216</td>
</tr>
<tr>
<td>3</td>
<td>0.999</td>
<td>0.01</td>
<td>0.282</td>
</tr>
</tbody>
</table>

As a last evaluation, we conducted a Monte Carlo simulation study to expose the relationship of mean value, variance and parameter \( \alpha \) on the expected error. All simulations are based on the average of 100,000 runs. As shown in Figure 8, the error increases slightly with the variance and decreases with the mean value. The depicted figures represent the approximation error for five and fifteen aggregated variables respectively. Another simulation run, depicted in Figure 6, shows the influence of a growing Service Value Network, i.e. a growing number of aggregation variables \( N \) for a typical setting with a mean value of 99.5\% and a variance of 0.001.

In this realistic scenario we can see, that the proposed approximation easily scales up to \( N = 30 \), meaning that we have 30 different Web service candidate pools, each of them possibly having many different concrete Web service instance alternatives. If
each candidate pool consisted of 10 different choices, one would have to optimize over $10^{30}$ composition alternatives, efficiently tractable using our proposed approximation scheme, yet only yielding an expected error value of 1.207%.

As stated in Section 4.1, the approximation can be further improved by adjusting the parameter $\alpha$. In Figure 7, one can see that parameter $\alpha$ can be effectively used to reduce the error even for low mean values, where a lower value of $\alpha$ is more appropriate when the QoS attributes are low as well.

6. Conclusion

In the preceding sections we described the importance of QoS aggregation in SVN’s, Web and Cloud service compositions. It became clear, that in both winner determination and other optimization approaches, computational efficiency is an important matter. Yet, current literature does not provide answers for efficiently aggregating multiplicative QoS attributes. We therefore proposed an approximation function, which is very suitable for high probability values close to one, as they are usually found for the availability and other probabilities in the context of Web or Cloud services.

A detailed evaluation explained different influences on the expected error, like the mean value, the variance or the introduction of a scoring function. In addition, we presented how parameter $\alpha$ within the approximation function can be used to achieve even more precise results.

Approaches as described in [18, 21, 25] can be of great benefit, as they enable a customized service configuration that maximizes the user experience by optimizing the resulting QoS according to the customer preferences. However, in any on-line scenario, these approaches are either limited to small problem sizes or restricted to additive QoS attributes, both resulting in sub optimal service allocations. Even in offline
scenarios, where computational time is less important, computational efficiency can be of great benefit, as computational time is costly or the model complexity can be increased in other aspect due to the greater efficiency of QoS aggregation. In this context our approach is striking as it provides a precise approximation scheme which eliminates the exponential problem complexity to polynomial time. Hence, computational effort can be reduced from minutes to milliseconds, which boosts winner determination and efficient allocation in SVN.

Our current approach is limited to values close to

![Figure 8: Stochastic error in %, qₖ ~ N(μ, σ²), N=5 (above) and N=15 (below)](image)
one. It will be interesting to find out, if similar or totally different linear approaches can be found for other domains.

From a practical point of view, an evaluation in an applied optimization scenario is planned, where the error of the approximation can actually be monetized and compared to the costs of an exact solution.

7. References


