A probabilistic approach to modeling and estimating the QoS of web-services-based workflows

San-Yih Hwang a,*, Haojun Wang b, Jian Tang c, Jaideep Srivastava b

a Department of Information Management, National Sun Yat-sen University, Kaohsiung 80424, Taiwan
b Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA
c Department of Computer Science, Memorial University of Newfoundland, St. John’s, Newfoundland, Canada A1B 3X5

Received 24 November 2006; received in revised form 3 April 2007; accepted 2 July 2007

Abstract

Web services promise to become a key enabling technology for B2B e-commerce. One of the most-touted features of Web services is their capability to recursively construct a Web service as a workflow of other existing Web services. The quality of service (QoS) of Web-services-based workflows may be an essential determinant when selecting constituent Web services and determining the service-level agreement with users. To make such a selection possible, it is essential to estimate the QoS of a WS workflow based on the QoSs of its constituent WSs. In the context of WS workflow, this estimation can be made by a method called QoS aggregation. While most of the existing work on QoS aggregation treats the QoS as a deterministic value, we argue that due to some uncertainty related to a WS, it is more realistic to model its QoS as a random variable, and estimate the QoS of a WS workflow probabilistically. In this paper, we identify a set of QoS metrics in the context of WS workflows, and propose a unified probabilistic model for describing QoS values of a broader spectrum of atomic and composite Web services. Emulation data are used to demonstrate the efficiency and accuracy of the proposed approach.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Web service QoS; Workflow QoS; Web service composition; Structural workflow; Web services; QoS aggregation

1. Introduction

Web services (WSs) have become a de facto standard for achieving interoperability among business applications over the Internet. In a nutshell, a WS can be regarded as an abstract data type that comprises a set of operations and data (or message types). Requests to and responses from WS operations are transmitted through SOAP (Simple Object Access Protocol), which provides XML-based message delivery over various types of transport protocols, including the popular HTTP connection. Although current Web services technol-
ologies have provided a fundamental architecture that enables the registry, discovery, and consumption of Web services, system integration requires a more complex process model to integrate the interactions of business applications. As a result, some standard bodies and vendors focus on the development of service composition languages, such as BPEL, OWL-S, ebXML, and WS Choreography, with the same aim of composing atomic Web services into workflows. In this work, we use the term “WS” to refer to a logical unit that comprises either a single WS operation (in the case of synchronous WSs) or a pair of invoke/receive operations (in the case of asynchronous WSs) and the term “WS workflow” to refer to a workflow composed of a set of WS invocations threaded into a directed graph. A WS workflow can in turn be wrapped as a WS, thereby enabling nested WS workflows.

With the help of above-mentioned technologies, a user’s task can now be performed by enacting a predefined WS. However, it is often not enough for the task to be just completed but it also has to be completed at some level of quality. The concept of QoS has been introduced and studied extensively in computer networks, multimedia systems, and real-time systems [2] as an overload management problem that measures non-functional aspects of the target system, such as timeliness (e.g., percentage of delayed messages) and completeness (e.g., percentage of dropped messages). QoS issue in WSs was discussed in [8] with emphasis on caching and replication. As WS technologies evolve, the issue of application level QoS guarantee for WS-workflow has attracted a new wave of interests from the research community due to its special characteristics. First, in a WS-workflow, the constituent WSs are loosely coupled, and there is no centralized middleware (like CORBA, TP-monitor, etc., in traditional workflows) used for coordinating the executions of the individual WSs. In other words, for each WS, all the related computations are wrapped inside, whether they are functional or not. This has the implication that the execution of a WS-workflow can be viewed as a collection of the executions, and therefore its QoSs can be ‘aggregated’ from the QoSs, of its constituent WSs. Second, in a traditional workflow, the constituent tasks and their execution engines are usually fixed at the design time. It is difficult, if not impossible, for an application to choose the executing engines for individual tasks dynamically toward meeting QoS requirements. On the contrary, the individual services in a WS-workflow can be bound to the service providers at run-time, and these service providers are usually heterogeneous and autonomous. In addition, a single service may have multiple choices for the service providers for the binding. This implies that an application can make a choice conveniently for the best bindings for the individual WSs so that the QoS of the entire WS-workflow can meet its expectation. In summary, while QoS is not a new topic, in the context of WS-workflow, some methodologies, such as QoS aggregation, which were previously of limited applicability, now becomes theoretically justifiable and practically significant.

Recently, many works have discussed how to specify and estimate the QoS of a WS-workflow effectively and efficiently based on the aggregations of the QoSs of its constituent tasks [4,11,16,27]. Almost all the existing works, however, have focused only on the static case by assuming that each WS has a deterministic QoS. The merit of this approach is simplification of the derivations. For example, consider a WS workflow $W$ that consists of three sequential WSs $W_1, W_2,$ and $W_3$ whose response times are 2, 3, and 4, respectively. It can be easily derived that the response time of $W$ is 9. We argue, however, that while assuming deterministic QoS values can simplify the computation task, it nonetheless has some limitations in applicability. This is because QoS measures of a WS are intrinsically probabilistic. For example, the response time for any invocation of a WS depends on such uncertain factors as the number of requests invoking it and the current load to the site where it is implemented. This is further complicated by the various types of information and data required for the execution of a WS. A WS-workflow embodies the data flow as well as control flow among the component WSs, and information is often streamed through services. While the input data of a WS could be in the form of simple scalar values produced by another WSs preceding the WS in the workflow, other more complicated data types, such as archival or real-time data, are also possible. Archival data may take more time to be shipped to the host site and real-time data are constantly changing, thereby inducing more uncertainties to the QoS measures of a WS. In fact, the QoS offered by a WS is heavily influenced by the loads that have been submitted to the service providers. These loads may require real time, interactive, or batch processing. If they require real-time processing, then the other requests will be affected. This situation requires the service implementation to adopt some contingent strategies so that the QoS requirements of the other non-real-time tasks
will not be unduly compromised. In the general case, the middleware must include an admission control mechanism which is responsible for managing the QoS requirements by the clients. Before accepting any request, the admission controller must assess the capability of the available resources to meet its QoS requirement. In our model, this is the probability distribution of the various QoS measures that it can offer. If by accepting the request, its remaining capability will be substantially down graded, it must republish this fact to the directory service.

In this probabilistic environment, the QoS of a composite WS cannot always be aggregated from its constituent WSs in a straightforward way. For example, consider two independent WSs $W_1$ and $W_2$, and a WS workflow $W$ comprising $W_1$ and $W_2$ with a parallel topology. In this case, $W$ is completed only when both $W_1$ and $W_2$ complete. Suppose we want to calculate the mean response time for $W$, given the mean response times for $W_1$ and $W_2$. It would be erroneous if we take the maximum of the means for $W_1$ and $W_2$ as the result. In fact, considering only the mean QoS values in a composite Web service, such as the work proposed in [27], is not enough when uncertainties are taken into account. In the context where we would like to study the QoSs using probabilistic terms, a simple approach would be to assume the QoS of every WS is normally distributed. This approach, however, is not realistic. First, not every QoS measure of an atomic WS follows a normal distribution. While some QoS measure such as response time can be reasonably described as a normal distribution, others, such as fidelity or reputation, are usually highly skewed and should not be modeled by normal distribution. Second, even if a QoS measure of each constituent WS is normally distributed, the QoS measure of the resultant WS workflow may not be normally distributed due to the various types of control constructs present in the WS workflow.

In our previous work [14], we have proposed a probability-based QoS model for atomic and composite WSs that views a QoS measure as a discrete random variable with a probability mass function (PMF), and described an efficient and accurate computation framework to derive QoS measures of a WS workflow. In this paper, we substantiate this model and the framework and perform a comprehensive evaluation on the proposed approach. The framework proposed in this paper allows users to verify a QoS objective of a newly developed WS workflow such as the following:

The probability that a WS workflow instance is completed in no more than 7 days is at least 0.95.

A QoS objective such as the one above can be easily verified once the PMF of the referred QoS is determined. For example, let $X$ be the random variable denoting the response time of the WS workflow. The above QoS objective can be verified by comparing $\Pr(X \leq 7)$ and 0.95. Our work is aimed to develop an approach to systematically estimating the PMF of a QoS measure of a given WS workflow from those of its constituent WSs. Such a QoS estimation may further serve as the basis for choosing a component WS among a set of candidate WSs (so as to achieve the desired QoS objective for the given WS workflow), called web services selection [20,22]. It can also be used to define a service-level agreement (SLA) between the WS workflow requester and the provider [30].

The main contributions of our research are as follows:

1. We identify a set of QoS metrics tailored for WSs and WS workflows, and provide an anatomy of these metrics.
2. We propose a probability-based QoS model that can be equally applied on both atomic WSs and WS workflows.
3. We develop a framework to derive a QoS measure of a WS workflow from those of its constituent WSs.
4. We explore alternative algorithms for computing the probability distribution functions of the WS workflow QoS. The efficiencies and accuracies of these algorithms are compared.

This paper is organized as follows. In Section 2 we analyze a set of proposed WS QoS metrics and define a probability-based QoS model to describe their values. In Section 3 we present the QoS computation framework for WS workflows. In Section 4 we describe algorithms for efficiently computing the QoS measures of a WS workflow. Section 5 presents the results of our performance evaluation, and Section 6 reviews related work. Finally, Section 7 concludes this paper and identifies directions for future research.
2. QoS model for WSs

2.1. WS QoS metrics

Many Web service QoS metrics have been proposed in the literature [4,11,16,22,23]. In our study, we broadly classify these metrics into the following six categories: performance, resources, dependability, fidelity, transactional properties, and security. Table 1 lists representative QoS metrics in each category and their meanings.

Note that some metrics are designed to directly measure the system capacity for executing a WS. Metrics used to measure the consumption of server resources, such as throughput, consumption of memory, CPU, and bandwidth, and time to repair, fall in the category of system-level QoS values. Often the system capacity for executing WSs (e.g., manpower for manual activities and computing power for automatic activities) are beyond the control of the invokers, are unlikely to be revealed due to autonomy considerations, and may change over time without underlying notifications. These metrics might be useful in some workflow context such as intraorganizational workflows (for determining the amount of resources to spend on executing workflows). However, for interorganizational workflows, where the execution of a WS may be controlled by another organization, system-level QoS metrics generally cannot be unilaterally measured. In fact, the system-level QoS values may directly impact some instance-level QoS measures such as the response time, the cost, the reliability, and the availability. The second QoS category is called service classes. QoS metrics in this category require all instances of the same WS to share the same values. Metrics of service classes include those categorized as transactional properties and security. In fact, these metrics may even be considered as functional features of a WS rather than QoS metrics, and are not discussed further. In this paper we focus on those WS QoS metrics that measure a WS instance and whose values may change across instances. These metrics, called instance-level QoS metrics, comprise response time, cost, reliability, availability, and fidelity rating. Note that cost is a complicated metric and could be a function of both service class and QoS value. For example, a WS that imposes weaker security requirements or incurs a longer execution time might be associated with a lower cost. Some services may adopt a different pricing scheme in which charges are based on factors other than usage (e.g., membership fee or monthly fee). In this paper, we consider the pay-per-service pricing scheme, which allows us to include cost as an instance-level QoS metric.

The five QoS metrics were adopted from the results of previous research into measuring the QoS of WS workflows [4,32]. However, we decided not to consider availability in our work because the availability of

<table>
<thead>
<tr>
<th>WS workflow QoS category</th>
<th>Instance-level QoS metrics</th>
<th>Service-classes metrics</th>
<th>System-level QoS metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Performance</td>
<td>Response time: Time elapsed from the submission of a request to the time the response is received</td>
<td>ACID properties</td>
<td>Throughput: The number of instances completed per time unit</td>
</tr>
<tr>
<td>2. Resources</td>
<td>Cost: The amount of money paid for executing an instance</td>
<td>Commit protocol (e.g., 2PC)</td>
<td>Memory/CPU/ bandwidth</td>
</tr>
<tr>
<td>3. Dependability</td>
<td>Reliability: The probability that the service can be completed successfully</td>
<td>Confidentiality</td>
<td>Time to repair</td>
</tr>
<tr>
<td></td>
<td>Availability: The probability that the service can be invoked successfully</td>
<td>Non-repudiation</td>
<td></td>
</tr>
<tr>
<td>4. Fidelity</td>
<td>Reputation rating: Usually measured as a scalar value, with a higher value being better</td>
<td>Encryption</td>
<td></td>
</tr>
<tr>
<td>5. Transactional properties</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Security</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
a WS workflow generally is not a function of the availabilities of its constituent WSs. Thus, our work considers four metrics (response time, reliability, fidelity rating, and cost) that can be equally applicable to both atomic WSs and WS workflows. These QoS metrics are defined such that different instances of the same WS may have different QoS values, and that the QoS of a composite WS can be derived from those of its constituent WSs.

2.2. Probabilistic modeling of WS QoS

We use a probability model for describing WS QoS. The QoS values of different (atomic or composite) WSs may follow different probability distributions (as already discussed in Section 1), and hence it is inadequate to assume a particular distribution for all QoS measures of WSs. To provide flexibility, we use a PMF on a finite scalar domain as our QoS probability model. In other words, each QoS measure of a WS is viewed as a discrete random variable whose PMF indicates the probability that the QoS measure assumes a particular value.

While it is natural to describe reliability, fidelity rating, and cost of a WS as discrete random variables and to model them as PMFs with domains of \{0 (fail), 1 (success)\}, a set of distinct ratings, and a set of possible costs respectively, PMFs may not seem natural for representing the response time whose domain is inherently continuous. Nevertheless, the response time can also be modeled as a discrete random variable if we observe it at a coarse granularity. Simple algorithms, such as dividing the range into subintervals with equal width or equal frequencies can be used to transform response time into a discrete attribute. More complex algorithms with different optimization purposes, e.g., classification [9] and association rule discovery [24] have also been available in the literature. As each discretization algorithm was invented with a purpose, in this work, we do not assume any particular algorithm for partitioning the range of response time into a set of subintervals. We simply view each subinterval as a representative number (e.g., the maximum) with a probability. For example, suppose that the probabilities of a WS being completed in subintervals \([0, 1], [1, 4], \text{and} [4, 7]\) are 0.2, 0.6, and 0.2 respectively. We can model the response time of that WS as a discrete random variable \(X\) with \(\text{Dom}(X) = \{0, 1, 4, 7\}\). The PMF of \(X\) is therefore represented as

\[
\begin{align*}
    f_X(0) &= 0 \\
    f_X(1) &= 0.2 \\
    f_X(4) &= 0.6 \\
    f_X(7) &= 0.2
\end{align*}
\]

As expected, finer granularity on the response time will yield a more accurate estimation at the expense of higher memory and computation overheads.

In real applications, requestors or providers of WSs may be more interested in knowing the probability of a range rather than a particular value with respect to a QoS measure. Such an inquiry can be answered by computing the cumulative distribution function (CDF). The CDF of a random variable \(X\), denoted as \(F_X(x)\), is defined as the probability that \(X\) is less than or equal to \(x\) for any \(x\). Let \(X\) be a discrete random variable with PMF \(f_X(\cdot)\). We can compute its CDF, \(F_X(x)\), by viewing it as a continuous random variable with probability density function \(p_X(x)\), defined as

\[
\begin{align*}
    p_X(x) &= 0 \quad \text{if } x \leq \text{Min(Dom}(X)) \\
    p_X(x) &= f_X(\text{ceiling}(x))/(\text{ceiling}(x) - \text{floor}(x)) \quad \text{if } \text{Min(Dom}(X)) < x \leq \text{Max(Dom}(X))
\end{align*}
\]

where \(\text{ceiling}(x) = \min\{y | y \in \text{Dom}(X) \text{ and } y \geq x\}\) and \(\text{floor}(x) = \max\{y | y \in \text{Dom}(X) \text{ and } y < x\}\).

The CDF \(F_X(x)\) can be defined as follows:

\[
\begin{align*}
    F_X(x) &= 0 \quad \text{if } x \leq \text{Min(Dom}(X)) \\
    &= \sum_{y \in \text{Dom}(X), y < x} f_X(y) + p_X(x) \cdot (x - \text{floor}(x)) \quad \text{if } \text{Min(Dom}(X)) < x < \text{Max(Dom}(X)) \\
    &= 1 \quad \text{if } x \geq \text{Max(Dom}(X)).
\end{align*}
\]
2.3. WS workflow composition

The QoS PMFs of a WS can be obtained from the QoS parameters listed in its profile if available, or derived from its past records of invocations. However, we need a way to determine the QoS PMFs of a newly developed composite WS that comprises a set of existing WSs. Different workflow composition languages may provide different constructs for specifying the control flow among constituent activities (e.g., see [28,29] for a comparison of the expressive powers of various workflow management systems and WS composition languages). Kiepuszewski et al. [17] defined a structured workflow model that consists of only the following four constructs that allows for recursive construction of larger workflows: sequence, or-split/or-join, and-split/and-join, and loop. The four constructs have been adopted in previous studies that used state charts for composing workflows [11], while an additional construct (called discriminator in [28] and fault tolerant in [4], as will be described below) was considered by Cardoso [4]. Although the structured workflow model is unable to model arbitrary workflows, it is nevertheless powerful enough to describe many real-world workflows and eases the job of workflow verification [17]. In this paper, we focus our attention on structured workflows, which dictates a hierarchical composition using a set of elementary constructs, referred to as structured constructs. Each structured construct organizes a set of constituent WSs in a unique way. The following lists five basic structured constructs:

1. **Sequence**: multiple WSs \(a_1, a_2, \ldots, a_n\) that are sequentially executed.
2. **Parallel split/join** (AND split/AND join): multiple WSs \(a_1, a_2, \ldots, a_n\) that can be executed concurrently and merged with synchronization.
3. **Exclusive choice** (exclusive split/exclusive join): multiple WSs \(a_1, a_2, \ldots, a_n\), among which only one WS can be executed.
4. **Discriminator** (AND split/OR join): multiple WSs \(a_1, a_2, \ldots, a_n\) that can be executed concurrently but merged without synchronization. That is, the first completed WS marks the completion of this construct and the other WSs will be ignored upon their completion.
5. **Loop**: an atomic or composite WS \(a\) guarded by a condition \(LC\), as shown graphically below (here \(a\) is executed iteratively until \(LC\) does not hold).

![Loop diagram](image)

The various standards/products for Web service composition include more constructs in addition to the five basic constructs described above. The work done by Jaeger et al. [16] examines all the workflow patterns summarized in [28] and identifies the following structured constructs in addition to the five basic constructs:

- **Interleaved parallel routing**: multiple WSs \(a_1, a_2, \ldots, a_n\) executed sequentially but in arbitrary order.
- **Multiple choice**: multiple WSs \(a_1, a_2, \ldots, a_n\) a subset of which can be concurrently executed.
- **m-out-of-n join**: used together with parallel or multiple choice and after the execution of \(n\) concurrent WSs, the subsequent WS can be invoked only when \(m\) WSs are completed.

These structured constructs cover most of the control constructs proposed in standards and products, including the leading proposals BPEL and OWL-S. As will be shown in the next section, the three additional constructs can be equivalently transformed to some of the five basic structured constructs in terms of QoS derivation.

---

1 OWL-S allows the QoS parameters of a published WS to be specified in its profile. The current WSDL standard, however, does not include the specification of QoS values. The QoS values of a WS could also be specified in its SLA.
3. Computing QoS measures of WS compositions

We now describe how to compute the QoS measures for each composition construct introduced in Section 2.3. Here we assume that the QoS values of each constituent WS of a composition construct are independent of those of the others. For example, the response time of a WS is assumed not to affect that of the next WS in a sequence construct. We consider that this is a reasonable assumption for an interorganizational WS workflow where different WSs may come from different organizations, which has also been assumed in previous studies [4,11,16,32]. We first show several basic operations on discrete random variables, including addition, multiplication, maximum, minimum, and exclusive selection. Based on these operations, we show how to compute the QoS PMFs for various composition constructs.

3.1. Basic operations for composing random variables

Let \( X \) be a discrete random variable with a finite sample space (or called domain) \( \text{Dom}(X) = \{x_1, x_2, \ldots, x_m\} \) and \( Y \) be an independent discrete random variable with a sample space \( \text{Dom}(Y) = \{y_1, y_2, \ldots, y_n\} \). The PMFs of \( X \) and \( Y \) are denoted as \( f_X() \) and \( f_Y() \), respectively. Let \( Z \) be the random variable generated by some operation on \( X \) and \( Y \). The domain and the PMF of \( Z \) are shown below.

3.1.1. Addition of two independent random variables (\( Z = X + Y \))

\[ \text{Dom}(Z) = \{z_1, z_2, \ldots, z_k\}, \text{Max}(m, n) \leq k \leq mn. \]

Each \( z_i, 1 \leq i \leq k, \) is the sum of some \( x \in \text{Dom}(X) \) and \( y \in \text{Dom}(Y) \).

\[ f_Z(z_i) = \sum_{x+y=z_i} f_X(x)f_Y(y). \]

The PMF of \( Z \) can be computed by first summing every element in \( \text{Dom}(X) \) and every element in \( \text{Dom}(Y) \), sorting these sums, and finally scanning through the sorted list to compute the PMF. The time complexity for the computation is \( O(mn \log(mn)) \).

3.1.2. Multiplication of two independent random variables (\( Z = X \cdot Y \))

\[ \text{Dom}(Z) = \{z_1, z_2, \ldots, z_k\}, \text{Max}(m, n) \leq k \leq mn. \]

Each \( z_i, 1 \leq i \leq k, \) is the product of some \( x \in \text{Dom}(X) \) and \( y \in \text{Dom}(Y) \).

\[ f_Z(z_i) = \sum_{x \cdot y = z_i} f_X(x)f_Y(y). \]

The computation is similar to that of addition, and the time complexity is thus \( O(mn \log(mn)) \).

3.1.3. Maximum of two independent random variables (\( Z = \max(X, Y) \))

\[ \text{Dom}(Z) = \text{Dom}(X) \cup \text{Dom}(Y). \]

\[ f_Z(z) = \begin{cases} f_X(z) \cdot \sum_{y \leq z \atop y \in \text{Dom}(Y)} f_Y(y) & \text{if } z \in \text{Dom}(X) \text{ and } z \notin \text{Dom}(Y); \\ f_Y(z) \cdot \sum_{x \leq z \atop x \in \text{Dom}(X)} f_X(x) & \text{if } z \in \text{Dom}(Y) \text{ and } z \notin \text{Dom}(X); \\ f_X(z) + f_Y(z) \cdot \sum_{x \leq z \atop x \in \text{Dom}(X)} f_X(x) & \text{if } z \in \text{Dom}(X) \text{ and } z \in \text{Dom}(Y). \end{cases} \]

The PMF of \( Z \) can be computed by sorting the elements in \( \text{Dom}(X) \cup \text{Dom}(Y) \), followed by scanning scanning through the sorted list to compute the PMF. The time complexity for the computation is \( O((m+n) \log(m+n)) \).
3.1.4. Minimum of two random variables \((Z = \text{Min}(X, Y))\)

\[
\text{Dom}(Z) = \text{Dom}(X) \cup \text{Dom}(Y).
\]

\[
f_Z(z) = f_X(z) \cdot \sum_{y \geq z \atop y \in \text{Dom}(Y)} f_Y(y) \quad \text{if } z \in \text{Dom}(X) \text{ and } z \notin \text{Dom}(Y);
\]

\[
f_Z(z) = f_Y(z) \cdot \sum_{x \geq z \atop x \in \text{Dom}(X)} f_X(x) \quad \text{if } z \in \text{Dom}(Y) \text{ and } z \notin \text{Dom}(X);
\]

\[
f_Z(z) = f_X(z) \cdot \sum_{y \geq z \atop y \in \text{Dom}(Y)} f_Y(y) + f_Y(z) \cdot \sum_{x \geq z \atop x \in \text{Dom}(X)} f_X(x) \quad \text{if } z \in \text{Dom}(X) \text{ and } z \in \text{Dom}(Y).
\]

The computation is similar to that of maximum, and the time complexity is thus \(O((m + n)\log(m + n))\).

3.1.5. Exclusive selection of \(X_1, X_2, \ldots, X_k\) with probabilities \(p_1, p_2, \ldots, p_k\) \((Z = \text{ES}_{1 \leq i \leq k}(X_i, p_i))\)

The exclusive selection, denoted \(\text{ES}_{1 \leq i \leq k}(X_i, p_i)\), where \(p_i, \ 1 \leq i \leq n\), is the probability that \(X_i\) is selected, is the exclusive selection of a random variable according to the associated probabilities. The result of \(Z = \text{ES}_{1 \leq i \leq k}(X_i, p_i)\) is a new random variable:

\[
\text{Dom}(Z) = \bigcup_{1 \leq i \leq k} \text{Dom}(X_i),
\]

\[
f_Z(Z = z) = \sum_{z \in \text{Dom}(X_i)} p_j \cdot f_{X_i}(z), \quad z \in \text{Dom}(Z).
\]

The PMF of \(Z\) can be computed by sorting the elements in \(\bigcup_{1 \leq i \leq k} \text{Dom}(X_i)\), followed by scanning the sorted list to compute the PMF. The time complexity for the computation is \((\sum_{i=1}^{k} n_i) \log(\sum_{i=1}^{k} n_i)\), where \(n_i\) is the size of \(\text{Dom}(X_i)\).

Example. Consider two random variables \(X\) and \(Y\) with \(\text{Dom}(X) = \{1, 2\}\) and \(\text{Dom}(Y) = \{10, 20\}\). Let \(f_X(1) = 0.3, f_X(2) = 0.7, f_Y(10) = 0.4,\) and \(f_Y(20) = 0.6\). The resultant random variable \(Z = X + Y\) is listed below:

\[
\text{Dom}(Z) = \{11, 12, 21, 22\},
\]

\[
f_Z(11) = 0.3 \times 0.4 = 0.12, \quad f_Z(12) = 0.7 \times 0.4 = 0.28,
\]

\[
f_Z(21) = 0.3 \times 0.6 = 0.18, \quad f_Z(22) = 0.7 \times 0.6 = 0.42.
\]

3.2. QoS measures of WS compositions

For each WS \(a\), we consider four QoS metrics, namely response time, cost, reliability, and fidelity, that are denoted by \(T(a), C(a), R(a),\) and \(F(a)\), respectively. A composition construct, denoted \(w\), involves several WSs. The QoS values of \(w\), denoted \(T(w), C(w), R(w),\) and \(F(w)\), can be computed from those of its constituent WSs through operations defined in Section 3.1, as shown below. Note that in our work each QoS measure is a random variable (with PMF) as opposed to a scalar value assumed by previous research [4,11,16,32], and the operations for composing random variables described in the previous subsection are needed for estimating the QoS measures of a composition construct. Besides, we consider only the normal execution of a composite WS and do not consider exceptions (e.g., see [5,7,15,18] for detailed discussion on workflow exceptions and transactions). The QoS computation in presence of exceptions requires a well-defined exception model, which is beyond the scope of this research.

(1) **Sequence**: \(w\) consists of a sequence of WSs \((a_1, a_2, \ldots, a_n)\).

**Cost**: The cost of \(w\) is the sum of the costs of its constituent WSs, i.e., \(C(w) = \sum_{i=1}^{n} C(a_i)\).

**Response time**: The response time of \(w\) is the sum of the response times of its constituent WSs, i.e.,
\[ T(w) = \sum_{i=1}^{n} T(a_i). \]

**Reliability:** The reliability of \( w \) is the product of the reliabilities of its constituent WSs, i.e.,
\[ R(w) = \prod_{i=1}^{n} R(a_i). \]

**Fidelity:** We assume that the fidelity of a sequential workflow is the weighted sum of the fidelities of its constituent WSs. The fidelity weight of each WS can be either manually assigned by the designer or automatically derived from past history, e.g., by using linear regression. That is,
\[ F(w) = \sum_{i=1}^{n} w_i F(a_i), \quad \text{where } w_i \text{ is the fidelity weight of } a_i. \]

(2) **Parallel split/join:** \( w \) consists of multiple concurrent WSs \((a_1, a_2, \ldots, a_n)\).

- **Cost:** The cost of \( w \) is the sum of the costs of its constituent WSs, i.e., \( C(w) = \sum_{i=1}^{n} C(a_i) \).
- **Response time:** The response time of \( w \) is the maximum of the response times of its constituent WSs, i.e., \( T(w) = \max_{i} \{T(a_i)\} \).
- **Reliability:** The reliability of \( w \) is the product of the reliabilities of its constituent WSs, i.e., \( R(w) = \prod_{i=1}^{n} R(a_i) \).
- **Fidelity:** The derivation is the same as that of the sequence construct, i.e., \( F(w) = \sum_{i=1}^{n} w_i F(a_i) \), where \( w_i \) is the fidelity weight of \( a_i \).

(3) **Exclusive choice:** \( w \) consists of a set of exclusive WSs \((a_1, a_2, \ldots, a_n)\), each associated with a probability \((p_i \text{ for } a_i)\), indicating the probability that \( a_i \) is executed.

- **Cost:** The cost of \( w \) is the exclusive selection of the costs of its constituent WSs with the associated probabilities, i.e., \( C(w) = \text{ES}_{1 \leq i \leq n}(C(a_i), p_i) \).
- **Response time:** The response time of \( w \) is the exclusive selection of the response times of its constituent WSs with the associated probabilities, i.e., \( T(w) = \text{ES}_{1 \leq i \leq n}(T(a_i), p_i) \).
- **Reliability:** The reliability of \( w \) is the exclusive selection of the reliabilities of its constituent WSs with the associated probabilities, i.e., \( R(w) = \text{ES}_{1 \leq i \leq n}(R(a_i), p_i) \).
- **Fidelity:** The fidelity of \( w \) is the exclusive selection of the fidelity of its constituent WSs with the associated probabilities, i.e., \( F(w) = \text{ES}_{1 \leq i \leq n}(F(a_i), p_i) \).

(4) **Discriminator:** \( w \) consists of a set of concurrent WSs \((a_1, a_2, \ldots, a_n)\) without synchronization.

- **Cost:** The cost of \( w \) is the sum of the costs of its constituent WSs, i.e., \( C(w) = \sum_{i=1}^{n} C(a_i) \).
- **Response time:** The response time of \( w \) is the minimum of the response times of its constituent WSs because the first completed WS marks the successful completion of \( w \), i.e., \( T(w) = \min_{i} \{T(a_i)\} \).
- **Reliability:** \( w \) fails only when all the constituent WSs fail. Thus, \( R(w) = 1 - \prod_{i=1}^{n} (1 - R(a_i)) \), where \( 1 - R \) denotes a complementary random variable of a reliability random variable \( R \). That is, for \( Z = 1 - R, f(Z = 1) = f(R = 0) \) and \( f(Z = 0) = f(R = 1) \).
- **Fidelity:** Note that the fidelity of the WS that is the first to complete becomes the fidelity of \( w \). Thus, the fidelity of \( w \) can be computed by the exclusive selection operation of fidelities of WSs with the probabilities that each WS is the first to be completed. Thus,
\[ F(w) = \text{ES}_{1 \leq i \leq n}(F(a_i), p_i(a_i)), \quad \text{where } p_i(a_i) = \prod_{k \neq i} P(T(a_k) > T(a_i)). \]

(5) **Loop:** A loop construct is defined as a repetition of a WS guarded by a condition LC. Cardoso et al. assumed that the number of iterations followed a geometric distribution [4]. However, the memoryless property of a geometric distribution fails to capture the common phenomenon that the repeated execution of an WS usually increases the probability of exiting the loop. Gillmann et al. [11] assumed the number of iterations to be uniformly distributed, which again will not hold in many applications. In this paper, rather than assuming a particular distribution, we simply regard the number of iterations as a PMF with a finite scalar domain.

Let \( f_{\text{PMF}}(l) \), \( 1 \leq l \leq c \), be the PMF of the number of iterations on an WS \( a \), where \( c \) is the maximum number of iterations.
Response time: If \( a \) is executed \( l \) times, the response time is \( T_a(l) = \sum_{1 \leq i \leq l} T(a) \). The response time of \( w \) is the exclusive selection on \( T_a(l) \) with probabilities \( f_{L(a)}(l) \), \( 0 \leq l \leq c \). Thus, the response time of \( w \) is

\[
T(w) = \text{ES}_{1 \leq i \leq c}(T_a(l), f_{L(a)}(l)).
\]

Cost: Let the cost of executing \( al \) times be \( C_a(l) \), i.e., \( C_a(l) = \sum_{1 \leq i \leq l} C(a) \). The cost of \( w \) is

\[
C(w) = \text{ES}_{1 \leq i \leq c}(C_a(l), f_{L(a)}(l)).
\]

Reliability: Let the probability of executing \( a l \) times be \( R_a(l) \). That is, \( R_a(l) = \prod_{1 \leq i \leq l} R(a) \), \( 1 \leq l \leq c \). The probability of \( w \) is \( R(w) = \text{ES}_{1 \leq i \leq c}(R_a(l), f_{L(a)}(l)). \)

Fidelity: When \( a \) is executed at least once, the fidelity of a loop structure is determined simply by that of its last execution. Thus, \( F(w) = F(a) \).

Table 2 summarizes the QoS measures of the five basic composition constructs.

### 3.2.1. Other constructs

The interleaved parallel route, supported by both BPEL and OWL-S, can be regarded simply as a sequence construct in terms of QoS derivation, and thus its QoS measures can be derived in the same way. The multiple choice, supported by both BPEL and OWL-S, can be regarded as a parallel split/join construct with slight modification on the PMFs associated with the QoS measures. Specifically, let \( w \) be a WS workflow involving a multiple choice consisting of a set of non-exclusive WSs \( \{a_1, a_2, \ldots, a_n\} \), each associated with an invocation probability \( (p_i \text{ for } a_i) \). Obviously, when a WS is not invoked, both of its cost and time can be accounted as 0 and its reliability be 1. That is, the domains of the new cost \( C \) and response time \( T \) for \( a_i \), denoted \( \text{Dom}(C(a_i)) \) and \( \text{Dom}(T(a_i)) \), becomes \( \text{Dom}(C(a_i)) \cup \{0\} \) and \( \text{Dom}(T(a_i)) \cup \{0\} \) respectively, and their PMFs are

\[
f_{C(a_i)}(c) = \begin{cases} p_i \cdot f_{C(a_i)}(c) & \text{if } c \in \text{Dom}(C(a_i)) \\ 1 - p_i & \text{if } c = 0 \end{cases}
\]

\[
f_{R(a_i)}(t) = \begin{cases} p_i \cdot f_{R(a_i)}(t) & \text{if } t \in \text{Dom}(T(a_i)) \\ 1 - p_i & \text{if } t = 0 \end{cases}
\]

The PMF for the reliability of \( a_i \) can be similarly derived as follows:

\[
f_{R(a_i)}(0) = p_i \cdot f_{R(a_i)}(0) \quad \text{and} \quad f_{R(a_i)}(1) = 1 - p_i + p_i \cdot f_{R(a_i)}(1).
\]

The cost, response time, and reliability of \( w \) are thus derived in the same way as in parallel split/join construct:

### Table 2

| QoS derivation of the five basic types of composition constructs with \( C(a), T(a), R(a) \), and \( F(a) \) denoting the cost, response time, reliability, and fidelity of an activity \( a \) |
|---|---|---|---|---|
| \( w = \text{Sequence}\left(a_1, a_2, \ldots, a_n\right) \) | \( \sum_{1 \leq i \leq n} C(a_i) \) | \( \sum_{1 \leq i \leq n} T(a_i) \) | \( \prod_{1 \leq i \leq n} R(a_i) \) | \( \sum_{1 \leq i \leq n} F(a_i) \) |
| \( w = \text{Parallel split/join}\left(a_1, a_2, \ldots, a_n\right) \) | \( \sum_{1 \leq i \leq n} C(a_i) \) | \( \max_{1 \leq i \leq n} \{T(a_i)\} \) | \( \prod_{1 \leq i \leq n} R(a_i) \) | \( \sum_{1 \leq i \leq n} W_{FS}(a_i) \) |
| \( w = \text{Exclusive choice}\left(a_1, a_2, \ldots, a_n\right) \) | \( \text{ES}_{1 \leq i \leq n}(C(a_i), p_i) \) | \( \text{ES}_{1 \leq i \leq n}(T(a_i), p_i) \) | \( \text{ES}_{1 \leq i \leq n}(R(a_i), p_i) \) | \( \text{ES}_{1 \leq i \leq n}(F(a_i), p_i) \) |
| \( w = \text{Discriminator}\left(a_1, a_2, \ldots, a_n\right) \) | \( \sum_{1 \leq i \leq n} C(a_i) \) | \( \min_{1 \leq i \leq n} \{T(a_i)\} \) | \( 1 - \prod_{1 \leq i \leq n}(1 - R(a_i)) \) | \( \text{ES}_{1 \leq i \leq n}(F(a_i)) \) |
| \( w = \text{Loop on } a \) | \( \text{ES}_{1 \leq i \leq n}(\sum_{1 \leq i \leq n} C(a_i), f_{L(a)}(l)) \) | \( \text{ES}_{1 \leq i \leq n}(\sum_{1 \leq i \leq n} T(a_i), f_{L(a)}(l)) \) | \( \text{ES}_{1 \leq i \leq n}(\prod_{1 \leq i \leq n}(1 - R(a_i)), f_{L(a)}(l)) \) | \( F(a) \) |

\( p_i \) is the chosen probability of \( a_i \), \( 1 \leq i \leq n \).

\( f_{L(a)}(l) \), \( 1 \leq l \leq c \), is the PMF of the number of iterations on the loop.
The fidelity weight of each WS has to be weighted by its execution probability. That is, the fidelity of $w$ is as follows:

$$F(w) = \frac{1}{\sum_{i=1}^{n} p_i \cdot w_i} \cdot \sum_{i=1}^{n} p_i \cdot w_i F(a_i).$$

Finally the $m$-out-of-$n$ construct, allegedly supported by OWL-S, can also be derived in a way similar to parallel split/join. The cost of $m$-out-of-$n$ is the same as that of parallel split/join. The response time, however, has to use another function $\text{Min-m}(\cdot)$, which returns the $m$th smallest of a set of random variables. $\text{Min-m}(\cdot)$ is a general function of $\text{Min}(\cdot)$ described in Section 3.1, and the formula is omitted here for brevity. However, the formula for $\text{Min-m}(\cdot)$ becomes complicated (and time-consuming to compute) when $m$ and $n - m$ are both large. Similar problems happen to the derivation of reliability and fidelity for $m$-out-of-$n$. Thus, deriving the probability QoS for a WS workflow involving $m$-out-of-$n$ construct can be practically done only when $m$ or $n - m$ is small.

The QoSs of WS workflows involving only structured constructs (as proposed in standards or products) can be recursively derived in a way describe above, whereas those involving non-structured constructs in general are difficult to estimate. One notable construct is link proposed in BPEL. A link is a construct designed for synchronizing two WSs located in any constructs (other than loops) in any depth, and thus its run-time behavior is difficult to predict. The QoS estimation of non-structured constructs is beyond the scope of this paper. In the following, we focus our attention on the five basic structured constructs as their QoS derivation procedure serves as the foundation for deriving the QoSs of other structured constructs.

4. Efficient QoS computation for a WS workflow

4.1. High-level algorithm

A structured WS workflow can be constructed recursively using the various structured constructs described in Section 3.2. Fig. 1a shows an example WS workflow associated with fulfilling orders for online stores. This simplified WS workflow, derived from TPC-W [26], is designed to process an order according to a user request. TPC-W is a transactional web service benchmark simulating the activities of a business oriented transactional application server and has been widely adopted by many vendors for evaluating their database and application servers. At the highest level, the WS workflow is a sequential construct that consists of order creation, stock payment check, and shipping arrangement. Stock payment check is a parallel split/join construct that comprises payment check and stock check. Payment check is in turn an exclusive choice construct composed of Paypal check and credit card check, whose selection is determined by the user as indicated in the order. Stock check is a loop construct on get item, which is executed for each ordered item. Each item will be acquired from the warehouse (get from warehouse) if it is in stock and from vendor (get from vendor) otherwise. The nested structure of the example WS workflow can be visualized as the tree structure shown in Fig. 1b.

The QoS of the entire WS workflow can be computed recursively; the pseudocode is listed in Fig. 2. Here we use $A$.type and $A$.QoS to denote the type (composite or atomic) and the set of QoS measures of a WS $A$ respectively. If $A$ is composite, $A$.construct and $A$.WSs are used to denote the construct name and the set of constituent WSs of $A$. Furthermore, $\text{SequenceQoS}(A$.WSs$)$, $\text{ParallelQoS}(A$.WSs$)$, $\text{ExclusiveQoS}(A$.WSs$)$, $\text{DiscriminatorQoS}(A$.WSs$)$, and $\text{LoopQoS}(A$.WSs$)$ are used to compute the QoS measures when $A$ is of type sequence, parallel split/join, exclusive choice, discriminator, and loop constructs, respectively. Their pseudocodes are obvious from the discussion in Section 3 and hence are omitted here for brevity.
4.2. Sample-space reduction

When combining PMFs of discrete random variables with respect to a given operation, the sample space of the resultant random variable may become huge. Consider adding $k$ discrete random variables each having $n$ elements in its respective sample space. The size of the sample space of the resultant random variable, in the worst case, is of the order of $n^k$. In order to limit the sample space of a PMF after each operation to a reasonable size, we propose grouping the constituent elements. Specifically, several consecutive scalar values in the sample space are represented by a single value, and the aggregated probability is computed. The problem is formally described below.

Let the domain of a random variable $X$ be $\{x_1, x_2, \ldots, x_s\}$, where $x_i < x_{i+1}$, $1 \leq i < s$, and let the PMF of $X$ be $f_X$. We assign another random variable $Y$ as an aggregate random variable of $X$ if there exists a partition $(j_1, j_2, \ldots, j_m)$ of $(x_1, x_2, \ldots, x_s)$, where $1 = j_1 < j_2 < \cdots < j_{m-1} < j_m = s$, such that

1. the domain of $Y$ is $\{y_r : y_r = x_{j_r}, \ 1 \leq r \leq m\}$, and
2. the PMF for $Y$ is...
The aggregate error of $Y$ with respect to $X$, denoted $\text{aggregate\_error}(Y, X)$, is the mean square error defined as

$$\text{aggregate\_error}(Y, X) = \sum_{r=2}^{m} \sum_{k=j_{r-1}+1}^{j_r} f_X(x_k) / f_Y(y_r) \cdot (x_k - y_r)^2.$$  

For example, consider the following PMF of a random variable $X$ with a domain $\{0, 1, 2, 3, 4, 5, 6, 7\}$: $f(X = 0) = 0$, $f(X = 1) = f(X = 2) = f(X = 3) = f(X = 4) = 0.1$, $f(X = 5) = f(X = 6) = f(X = 7) = 0.2$.

For a partition $(0, 3, 5, 7)$, the PMF of the corresponding random variable $Y$ is

$$\text{Dom}(Y) = \{0, 3, 5, 7\};$$

$$f(Y = 0) = 0, \quad f(Y = 3) = 0.3, \quad f(Y = 5) = 0.3 \text{ and } f(Y = 7) = 0.4.$$ 

And the aggregate error of $Y$ with respect to $X$, $\text{aggregate\_error}(Y, X)$, is 2.5.

The aggregate random variable discovery problem (ARVD)

The concept of $\text{aggregate\_error}$ introduced above measures the extent to which the aggregate random variable differs from the original random variable. Therefore, it should be minimized. For this purpose, an ARVD problem is defined as follows.

Given a random variable $X$ of domain size $s$ and a desired domain size $m$, the problem involves finding an aggregate random variable $Y$ of domain size $m$ so as to minimize its aggregate error with respect to $X$.

4.2.1. Dynamic programming method

An optimal solution to this problem can be obtained by formulating it using dynamic programming. Let $e(i, j, k)$ be the optimal aggregate error of partitioning $x_i, x_{i+1}, \ldots, x_j$ into $k$ subsequences. This produces the following recurrence:
\[e(i, j, k) = \min_{a < j, b < k} \{e(i, a, b) + e(a + 1, j, k - b)\} \quad \text{if} \quad j - i + 1 > k \text{ and } k > 1;\]
\[e(i, j, k) = 0 \quad \text{if} \quad j - i + 1 = k;\]
\[e(i, j, 1) = \text{error}(i, j),\]

where \(\text{error}(i, j)\) is the aggregated error introduced by representing \((x_i, x_{i+1}, \ldots, x_j)\) as a single value \(x_j\). Specifically, \(\text{error}(i, j) = \sum_{k < j} \frac{f_X(x_k)}{\sum_{i \leq k < j} f_X(x_i)} (x_k - x_j)^2\).

The time and space complexities of the dynamic programming algorithm are \(O(s^3m^2)\) and \(O(s^2m)\), respectively. Suppose we add two random variables each with a sample-space size \(m\), where we would like the size of the sample space of the resultant aggregate random variable to be reduced to \(m\). In this case, the time and space complexities of the dynamic programming method become \(O(m^3)\) and \(O(m^3)\), respectively.

### 4.2.2. Greedy method

We propose a heuristic method for reducing the computation overhead associated with solving this problem. The idea is to continuously merge adjacent pairs of samples that give the lowest error until the desired sample-space size is reached. When an adjacent pair \((x_i, x_{i+1})\) is merged, \(x_i\) is eliminated and the probability of \(x_{i+1}\) is changed as
\[f_X(x_{i+1}) = f_X(x_i) + f_X(x_{i+1}).\]

The error of merging \((x_i, x_{i+1})\), denoted \(\text{pair\_error}(x_i, x_{i+1})\), is computed as
\[\text{pair\_error}(x_i, x_{i+1}) = \frac{f_X(x_i)}{f_X(x_i) + f_X(x_{i+1})} (x_i - x_{i+1})^2.\]

We use a priority queue to store the error of merging each adjacent pair. In each iteration we perform the following steps:

1. Extract an adjacent pair with the least \(\text{pair\_error()}\) value from the priority queue, say \((x_i, x_{i+1})\).
2. Remove \(x_i\) and \(x_{i+1}\) from the sample space \(X\) and insert a new element \(x' = x_{i+1}\), where \(f_X(x') = f_X(x_i) + f_X(x_{i+1})\).
3. Compute \(\text{pair\_error}(x_{i-1}, x')\) if \(i > 1\) and \(\text{pair\_error}(x', x_{i+2})\) if \(i < n - 1\). Delete \(\text{pair\_error}(x_{i-1}, x_i)\) and \(\text{pair\_error}(x_{i+1}, x_{i+2})\) from the priority queue, and insert \(\text{pair\_error}(x_{i-1}, x')\) and \(\text{pair\_error}(x', x_{i+2})\) into the priority queue.

In each iteration, steps 1 and 3 take \(O(\log(s))\) time while step 2 takes a constant time. The total number of iterations is \(s - m\). Thus, the time complexity of this greedy approach is \(O(s\log(s))\).

When combining the QoS measures of \(n\) WSs, we can perform pairwise QoS combinations \(n - 1\) times and apply a sample-space reduction method as described above after each combination. Suppose each random variable has the same sample-space size \(m\). The addition of two random variables can result in a new random variable with a sample-space size up to \(m^2\). We then apply some sample-space-reduction technique to reduce the sample-space size down to \(m\) before combining the next random variable.

### 4.3. Analysis of time complexity

Of the five basic operations on random variables described in Section 3.1, the addition and the product operations incur the largest time complexity. In addition, the \(i\)th loop structure with a maximum of \(c_i\) iterations needs \(O(c_i)\) addition operations. All other structures with \(n_j\) WSs need a maximum of \(n_j\) addition operations. Thus, the total number of addition operations required to compute one QoS measure is \(O(\sum n_j + \sum c_i) = O(n + c)\), where \(n\) is the total number of WSs in the WS workflow and \(c\) is the summation of the maximum iteration times of all loop structures. Suppose the sample-space size for each random variable is \(m\) and after each addition operation the size of resultant random variables is reduced to \(m\) again. Considering that the time complexity of each addition operation is \(O(m^2\log(m^2)) = O(m^2\log(m))\) and using a greedy approach for reducing the sample-space size from \(n_2\) to \(m\) requires \(O(m_2\log(m))\), computing the PMF of one QoS measure takes \(O((n + c)m^2\log(m))\) time. Assuming that \(c\) and \(m\) are constants, the execution time of the
greedy method for computing QoS measures of a WS workflow increases linearly with the number of WSs in the WS workflow.

5. Performance evaluation

In this section, we report our experimental results obtained by evaluating the performance of the proposed probabilistic QoS computation. Our experimentation was conducted on a PC server equipped with an Intel P4 2.66-GHz CPU and 512 MB of RAM. WSs were deployed on Weblogic 8.1 application server, and WS workflows were implemented using the Oracle BPEL Process Manager 10. The QoS measures for each WS followed a certain probability distributions as given by our experimental parameters. Each execution of a WS would generate QoS values for response time, cost, reliability, and fidelity according to the given distributions. Each WS was executed 10,000 times and the histogram for each QoS metric serves as the PMF of its QoS measure. Based on the QoS PMFs of each component WS, we applied our QoS computation framework to compute the QoS PMFs of a target WS workflow.

5.1. Comparison of two methods for reducing the size of the sample space

The first set of experiments was designed to show how the two proposed techniques for reducing the size of the sample space, namely dynamic programming and the greedy method, impact the accuracy of the resultant PMFs. Consider a simple WS workflow that consists of only two sequential, independent WSs. Let $Z$ be the response time of the WS workflow, and $X$ and $Y$ be the response times of the two WSs. Obviously $Z = X + Y$.

For ease of comparison, we assumed that the generative models for both $X$ and $Y$ are normal distributions, which allows us to theoretically compute the generative model of $Z$. Table 3 lists the parameters for this set of experiments.

The CDF is used as the performance metric. Fig. 3a shows the CDFs obtained by using dynamic programming and the greedy method, with the theoretical generative model serving as the benchmark. The figure indicates that the CDFs obtained from dynamic programming and greedy methods are very close to the theoretical CDF. The subtle differences between the CDF obtained from each method and the theoretical CDF are revealed in Fig. 3b. Although the differences are indeed very small (less than 0.01), it can still be seen that dynamic programming in general leads to better CDFs with the mean probability error being 0.001494, compared to 0.002136 for the greedy method.

We next vary the size of the sample space of the aggregate random variable $Z$ and compare the mean square errors incurred by dynamic programming and the greedy method. Fig. 4 shows the experimental results. As expected, a larger sample space leads to a smaller mean square error. Moreover, the greedy method consistently incurs a slightly higher mean square error for different sample-space sizes.

As demonstrated by the time-complexity analysis presented in Section 4.2, the running time increases much more rapidly for the dynamic programming method than for the greedy method, whereas this experimental set indicates that the improvement in accuracy is only small. Therefore, in the following experiments we only considered the greedy method because it is efficient and achieves an accuracy that is close to optimum.

5.2. Evaluation of QoS measures of a WS workflow

The second set of experiments aims to measure the accuracy of the QoS PMFs computed using the proposed methods. We implemented the example WS workflow shown in Section 4.1, namely order fulfillment,

Table 3
Parameters of experiment set 1

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generative model of $X$</td>
<td>$N(100,10)$</td>
</tr>
<tr>
<td>Generative model of $Y$</td>
<td>$N(90,20)$</td>
</tr>
<tr>
<td>Generative model of $Z$</td>
<td>$N(190,10\sqrt{5})$</td>
</tr>
<tr>
<td>Sample-space size of $X$, $Y$</td>
<td>20</td>
</tr>
<tr>
<td>Sample-space size of $Z$</td>
<td>30</td>
</tr>
</tbody>
</table>
on our experimental platform. The generative model for the response time of each atomic WS is a normal distribution with minimum- and maximum-value constraints. The parameters are listed in Table 4. The probabilities that the loop structure (GetItem) will be executed 1, 2, 3, 4, and 5 times are 0.1, 0.2, 0.4, 0.2, and 0.1, respectively.

The benchmark of this experiment was obtained through emulation on our experimental platform. The emulation was executed 10 times, with each emulation generating 1 million instances. A PMF was derived for each execution, and the average of the 10 PMFs served as the benchmark. The emulation result was verified as follows. Each run of the emulation results in a response time PMF whose domain contains 100 equally spaced values. Thus, each sample value is associated with 10 probabilities obtained from the 10 executions. The standard deviation of these 10 probability values was then calculated. It was found that the average and maximum of the 100 standard deviations were very small, and the 99.7% confidence interval for each sample value is no larger than \([\mu - 0.002, \mu + 0.002]\), where \(\mu\) is the mean. This result demonstrates that the

---

**Fig. 3.** (a) CDFs of dynamic programming and the greedy method with the theoretical method serving as the benchmark and (b) differences in CDFs of each method relative to the theoretical method.

**Fig. 4.** Mean square errors of dynamic programming and the greedy method for different sample-space sizes.
emulation results were indeed very close to the real ones. Using the PMF generated by emulation as the benchmark, Fig. 5 illustrates the difference between the cumulative probabilities computed using our approach incorporating the greedy method for sample-space reduction and the benchmark, when the aggregate size of the sample space is set to 30. It can be seen that the difference is very small, with mean difference less than 0.001.

Similar experiments were conducted for the cost and fidelity. The results exhibit trends similar to that of the response time and are omitted here for brevity. Note that computing the reliability PMF of any composition construct does not increase the sample size, and thus there is no need to test techniques for reducing the size of the sample space with respect to the reliability metric.

6. Related work

This paper focuses on the QoS estimation of structured workflow. There have been several formal process models proposed in the literature, such as FSM [3], Petri-Net [1], and event calculus [6], that can express more (rare) types of control constructs. Despite their superior expressive power, it is difficult to derive the semantics of control constructs expressed in these models, thereby jeopardizing QoS estimation.

QoS-based WS selection has attracted much attention in recent years. Previous work has attempted to optimize the selection for a single activity, with the most recent work focusing on the selection of WSs in order to satisfy the QoS requirements of a WS workflow. Wang et al. applied fuzzy logic for selecting a WS for a given task that meet an individual’s personal needs [27]. Patel et al. [22] advocate the selection of the best-\(m\) WSs for each activity, and these WSs are then randomly assigned to activity instances according to their fitness values as derived from their QoS values. It was demonstrated using real WSs that selecting multiple WSs for a single activity tends to improve the load balancing. Zeng et al. [32] identified a set of QoS metrics and proposed the
application of linear programming to the selection of the execution plan with the optimal QoS value. However, they considered only limited workflow constructs that did not include looping and fault tolerance. Yu et al. [31] proposed a scheme to optimize the end-to-end QoS for sequential structure. They use a utility function that is an increasing function of the QoS quality. The optimization problem is formulated as a knapsack problem. All these work considers only deterministic QoS values.

Some work concentrates only on a single QoS measure. Menasce [19] proposed a scheme to estimate the throughput of a composite WS from those of its constituent WSs, which serves as a basis for selecting WSs. Grassi et al. [12] proposed a framework to recursively aggregate the reliability of a composite WS-workflow based on the constituent composite or atomic WS-workflows. Each sub-WS-workflow is either no, or partial, or total-transparent, corresponding respectively to high, medium and low autonomy, and revealing no, partial and all the information on its internal structure.

Our work is closest to the workflow reduction technique proposed in [4] and [16], which iteratively applies a set of reduction rules until a single workflow is reached. The work reported by Jaeger et al. [16] extends the work proposed in [4] by considering more control constructs and computing both maximum and minimum values for each QoS measure. Although Cardoso et al. mentioned the possibility of deriving distribution functions for QoS of workflow tasks, the proposed reduction rules were applied to compute only deterministic QoS values. In contrast, our probability-based QoS model enables the estimation of the probability distribution functions for the QoS values of a given workflow, which provides broader applicability.

The only work we know of at this time that uses probabilistic method for aggregating QoS parameters is in [1]. The authors use a discrete-time stochastic Petri-Net model to aggregate QoS of a composite structure from the QoSs of its components. The discussion there, however, is restricted only on throughput. Also, it is not clear how the algorithm mentioned there reduces the sample space in order to achieve efficiency.

There have been several previous proposals of using simulation to measure the QoS of a workflow [4,11,25], with a major focus on timeliness measures. The simulation approach models a workflow as a queuing system with transition probabilities on conditional branches. However, the simulation approach usually incurs high computation overhead and may result in an overkill for QoS estimation.

The critical-path method (CPM) and the program-evaluation review technique (PERT) are project management techniques for estimating the execution time and identifying the bottlenecks of a project [13]. The key idea of CPM/PERT is to identify the critical path, which is the sequence of stages determining the minimum time needed to complete the entire project. In CPM, the time of each activity is deterministic. PERT assumes a beta distribution and independence of the execution time of each activity. It uses three parameters (the most-optimistic, most-likely, and most-pessimistic estimates) to determine the mean value ($\mu$) and standard deviation ($\sigma$) for the execution time of an activity according to

$$\mu = (\text{most optimistic} + (4 \times \text{most likely}) + \text{most pessimistic})/6,$$

$$\sigma = (\text{most pessimistic} - \text{most optimistic})/6.$$

The CPM/PERT network is much simpler than the structure of business processes, and does not involve exclusive choice and loop structures. It is more suitable for describing manufacturing processes, whose component activities are usually deterministic. CPM/PERT focuses on the computation of process time and the tradeoffs between the time and cost, whereas other QoS metrics, such as fidelity and reliability, need to be supported in WS workflows.

A notable research that has attracted much attention recently is automatic Web service composition. The automatic WS composition problem involves verifying the feasibility of composing a target WS using a community of (atomic) Web services. Each (composite or atomic) WS is associated with a formal model that constrains the order of its constituent operations. Notable models include Petri Net [21] and Finite State Machine [3,10]. While each operation in the composite WS can be delegated to any one of the candidate operations contained in several WSs in the WS community, all constraints, as imposed by the respective formal models, have to be satisfied. The automatic Web service composition is indeed an important problem and should be the first to be performed to identify a set of feasible WSs for composing a target WS. The probabilistic QoS computation framework proposed in this paper can be subsequently applied to further give the QoS values of the target WS.
7. Conclusions

In this paper, we have analyzed the QoS metrics for WSs and proposed a probability-based QoS model. A QoS measure of an atomic or composite WS is quantified as a PMF. We have described algorithms to compute the QoS measures of a WS workflow from those of its constituent WSs. We have introduced the problem of computing the least-error QoS PMF when composing a WS workflow, and showed that a very large search space is required. We have provided a dynamic programming formulation for the optimal solution and an efficient approximation heuristic. The proposed solutions have been evaluated via synthetic data generated by our experimental platform, and the experimental results show that our proposed framework achieves high accuracy and is computationally efficient.

This work can be extended in several directions. First, our work does not address the dependences on the selection of a WS and its QoS measures. In fact, there may exist dependencies on the proposed QoS measures, e.g., slower response time may infer lower cost. A possible approach for describing the dependencies on QoS measures and the WS selection is to use Bayesian networks. Second, the online monitoring on the execution of WS workflows needs investigation. The goal is to meet the online requests of users about the QoS of an executing WS workflow, which may differ from the estimations made at design time, and/or to satisfy user-defined QoS objectives. This service will alert the requester or the provider of a WS workflow instance at an early stage about the possible violation of some QoS objective. Finally, we plan to integrate QoS model into some formal model such as Petri Net, Finite State Machine, or Pi-Calculus. Although many business processes can be specified using the structured workflow model, its expressive power is not complete. Formal models have sufficient expressive power and are equipped with strong analysis capability. We are currently exploring the WS selection problem based on Finite State Machine by considering QoS.

Acknowledgement

This work was partially supported by “Aim for the Top University Plan” of the National Sun Yat-sen University and Ministry of Education, Taiwan, ROC.

References


