A Tree-Based Reliability Model for Composite Web Service with Common-Cause Failures

Bo Zhou¹, Keting Yin¹, Shuai Zhang¹, Honghong Jiang¹, and Aleksander J. Kavs²

¹ College of Computer Science and Technology, Zhejiang University, Hangzhou, China
(bzhou,yinketing,zhangshuai,zhejianghonghong)@zju.edu.cn
² State Street Corporation, Boston, MA, USA
ajkavs@statestreet.com

Abstract. Reliability is one of the most important quality dimensions for web services. Current reliability models for composite web service assume the statistical independence of its component web services, which is often not the case. In this paper, we research on the reliability correlation of component web services by identifying common-cause failures (CCF) and propose a tree-based reliability model for composite web service. We also present the method to estimate overall reliability of composite web service based on our reliability model. Evaluation shows that our method can improve the accuracy of reliability estimation for composite web service and is particularly valuable when designing fault-tolerant service-based system.

Keywords: composite web service, reliability model, common-cause failure.

1 Introduction

With the prevalence of web services [1] in recent years, more and more applications are developed based on Service-Oriented Architecture (SOA) [2]. Compared with the traditional software systems, service-centric systems are built upon bundles of web services, which could be developed by their own, or more commonly, provided by external parties and accessed over the Internet. Thus, the runtime environment for the web services-based system is prone to failure due to the uncertainty in the Internet and external parties. Meanwhile, business level applications have very stringent requirements on the reliability, for example, 99.99% requests should be processed correctly in a reasonable time. Not surprisingly, reliability of web services has been researched extensively and is still a hot topic.

Reliability is often defined as "the probability that it successfully responds within a reasonable period of time." [3]. For atomic web service, it can be measured using statistical method based on the history logs; while for composite web service, its reliability depends on the reliabilities of the underlying component services. Various methods has been applied to estimate the aggregated reliability for the composite web service, such as reduction method [4] [5] and Petri Net [6].

However, as far as the authors know, existing methods for reliability modeling of composite web services often assume the statistical independence between the component web services. While in real world environment, this assumption could be
often violated. Service providers tend to publish a series of web services that are closely related and they are potential candidates to be selected for the same web service composition; moreover, service consumers are likely to choose several component services from the same provider for reasons such as package discounts or trust in particular providers. If multiple component services are from the same provider or even deployed on the same server, accidents like system crash or regional power outage will lead to simultaneous failing of these services. Such correlations between component services are called Common Cause Failures (CCF) [7], which should be identified and modeled properly in order to estimate the reliability of composite web service more accurately, especially when developing fault-tolerant systems.

In this paper, we investigate the category of CCFs in the context of composite web service and propose a novel tree-based reliability model for composite web service that considers the presence of CCFs. We also present a recursive algorithm to compute the aggregated reliability. Our reliability model is more accurate and conforms to the reality better.

The rest of our paper is structured as follows. Section 2 introduces the preliminary knowledges. Section 3 presents our tree-based reliability model and the computing algorithm. Applications are demonstrated Section 4. Section 5 is the related work. Conclusion and future works are discussed in Section 6.

2 Preliminaries

Table 1 summarizes the reliability aggregation rules for four major composition patterns borrowed from existing literatures [8]. For sequence and parallel patterns, the aggregated reliability of the both is computed as the product of its component services' reliabilities.

<table>
<thead>
<tr>
<th>Composition Pattern</th>
<th>Reliability Aggregation Rule</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>$\prod_{i=1}^{n} \text{Rel}(w_i)$</td>
<td>$n$: number of services in the pattern.</td>
</tr>
<tr>
<td>Parallel</td>
<td>$\prod_{i=1}^{n} \text{Rel}(w_i)$</td>
<td>$n$: number of services in the pattern.</td>
</tr>
<tr>
<td>Conditional Loop</td>
<td>$\min{\text{Rel}(w_1), ..., \text{Rel}(w_n)}$</td>
<td>Each service is on a branch in the pattern.</td>
</tr>
<tr>
<td></td>
<td>$\text{Rel}(w_i)^k$</td>
<td>$k$: average number of cycles in the loop.</td>
</tr>
</tbody>
</table>

As we stated above, these aggregation rules only hold when the component services are statistically independent, but in real world, it is quite possible that some component services are correlated with each other in certain way. We observe that a web service could fail due to reasons in different levels. Table 2 categorizes the failures of web services into four levels and presents possible scenarios for each level. The listed scenarios are not comprehensive but should be sufficient for further discussions.
### Table 2. Failures in different levels for Web Service

<table>
<thead>
<tr>
<th>Level</th>
<th>Possible Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>logical errors in the implementation, binding fault, incompatibility issues</td>
</tr>
<tr>
<td>Host</td>
<td>system crash, unresponsiveness due to server overload, hardware failure</td>
</tr>
<tr>
<td>Provider</td>
<td>regional outage, closure of the operating company</td>
</tr>
<tr>
<td>Network</td>
<td>communication congestion, hardware failure</td>
</tr>
</tbody>
</table>

Service level failures will only affect a specific web service, i.e., failings of component services in this level are independent. Meanwhile, failures in the other three levels could cause multiple web services fail simultaneously, which are CCFs of these failed web services. Furthermore, service, host, provider and network are a list of levels of increasing coarse granularity, which can be organized using a tree structure:

### Definition 1. Reliability Hierarchy Tree (RHT)

A Reliability Hierarchy Tree is used to model the reliability hierarchy for web services. It is a tree with four levels. We formally define it as $RHT = (V, L, C, \delta)$, where

- $V$ is the set of all nodes in the RHT. It could represent a network, provider, host machine or web service depending on its located level.
- $L = \{N, P, H, WS\}$ is the set of levels in the RHT, which are network (N), provider (P), host (H) and service (WS) levels respectively from top (root) to the bottom (leaf). $\forall v \in V$, we use $L(v)$ to denote its level.
- $C = D \subseteq V \times V$ is the set of parent-child relations, where $\forall (x, y) \in C$, $x$ is the parent node and $y$ is the child node. Parent-child relations are identified by the following rules: (1) if a provider $p$ is located in the network $n$, then add $(n, p)$ to $C$; (2) if a host machine $h$ is maintained by the provider $p$, then add $(p, h)$ to $C$; (3) if a web service $ws$ is deployed on the server $h$, then add $(h, ws)$ to $C$.
- $\delta = V \times R^+$ associates each node with a positive real number denoting its reliability value. $\forall v \in V$, we use $\text{Rel}(v)$ to denote its reliability value.

Fig. 1 is an example of RHT. For the end user, the reliability of the web service is an integrated value of all levels. We use $\text{RelU}(ws_i)$ to represent the web service reliability from the user's perspective. For a single web service, its reliability from user perspective could be calculated easily given the RHT:

$$\text{RelU}(ws_i) = \text{Rel}(ws_i) \times \text{Rel}(H_j) \times \text{Rel}(P_m) \times \text{Rel}(N_n) \tag{1}$$

where $H_j$, $P_m$, $N_n$ are corresponding host, provider and network for this service.

![Fig. 1. An example of Reliability Hierarchy Tree](image-url)
3 Tree-Based Reliability Model

In this section, we present the reliability model and aggregation method for the composite web service with common-cause failures. Four common composition patterns are discussed, which are sequential, parallel, conditional and loop.

3.1 Sequential Pattern

The sequential pattern is the most basic and common means to compose the web services, thus we use it as the starting point to present our method.

Definition 2. RelTree

RelTree is used to model the reliability relations of the component services in web service composition. Formally, given a composite web service \( cws = (ws_1, ws_2, \ldots, ws_n) \), its \( \text{RelTree} = (V, L, C, \delta) \) is a consolidated RHT where,

\[
\forall v \in V \text{ and } L(V) = WS \Rightarrow \exists ws \in cws, \text{ where } ws \text{ is represented by the node } v;
\]

\[
\forall ws \in cws \Rightarrow \exists v \in V, \text{ where } L(V) = WS.
\]

Definition 3. RelForest

The root node of RelTree represents a network. If a composite web service contains services from multiple networks, then its CCF-aware reliability model will contain several RelTree, which is called RelForest. Without loss of generality, we assume component services of a composite web service locate in only one network in the rest of this paper for simplicity.

RelTree can be constructed from RHT by pruning unrelated nodes and branches. Algorithm 1 shows the process.

---

**Algorithm 1. RelTree Building**

**Input:** RHT (denoted as \( rh \)), composite web service (denoted as \( cws \))

**Output:** RelTree for \( cws \)

1. **foreach** component web service \( ws \) in \( cws \)
2. find the corresponding node \( s \) in \( rh \);
3. **do**
4. mark \( s \) as visited;
5. **if** ( \( s \) is root) **then** **break**;
6. set \( s = s.\text{parent} \);
7. **if** ( \( s \) is visited) **then** **break**;
8. **repeat** from line 3;
9. **end for**
10. prune all the unvisited node;
11. **return** resulting RelTree;

---

Fig. 2 is an example of RelTree constructed from RHT, where the composite web service \( cws = \{ws_1, ws_4, ws_5\} \). Each node in the resulting RelTree is annotated with the reliability value.

With the RelTree, we can answer the following questions about the reliability aspects of the composite web service accurately:
• Q1: What is the reliability value of a component web service?
• Q2: What is the reliability value of the composite web service ignoring the CCFs?
• Q3: What is the reliability value of the composite web service when CCFs are considered?
• Q4: What is the probability that no more than n services failed simultaneously in the composition?

The answer for Q1 has been discussed above and the answer for Q2 is also straightforward which is the product of reliability of each component web service as calculated in Q1. Q3 can be reduced to Q4 when \( n = 0 \) (i.e. all component web services are in good condition). Therefore, we give Algorithm 2 as the solution to Q4.

**Algorithm 2. RelTreeCal**

**Input:** the RelTree or subTree of RelTree for the composite web service (denoted as relTree), maximum number of failed nodes (denoted as \( n \), and \( n \) should not be larger than the number of leaves in the tree)

**Output:** the probability that no more than \( n \) services fail simultaneously in the service composition

```plaintext
1  initialize rel ← 0;
2  foreach m from 0 to n
3    initialize \( t \) ← number of leaves in the relTree, \( v \) ← root node, \( p \) ← 0;
4    if \( v \) is a leaf
5      \( p = (m == 0) \) ? Rel(\( v \)) : 1 − Rel(\( v \));
6    else
7      set \( fn \) ← first child node of \( v \),
8      set \( on \) ← the set of child nodes of \( v \) except \( fn \),
9      set \( a \) ← number of leaves in the subTree \( fn \), \( b \) ← \( t \) − \( a \),
10     set \( min \) ← \( (b >= m) \) ? 0 : \( (m - b) \), \( max \) ← \( (a >= m) \) ? \( m \) : \( a \);
11     \( p = \sum_{i = min}^{max} pf * po \), where
12     \( pf = \text{RelTreeCal} (fn, i), po = \text{RelForest} (on, m - i) \);
13   end if
14  set rel ← rel + p;
15 end for
16 return rel;
```
The main steps of the algorithm are explained as follows:

- **Line 4, 5:** If \( v \) is a leaf, simply set its reliability or failure value depending on required number of failed node

- **Line 6-13:** Store first child node of \( v \) into \( fn \) and the rest child nodes (if any) into \( on \), initialize \( a, b \) as the number of leaves in \( fn \) and \( on \) respectively, set \( min, max \) as the least and maximum number of failed services in the subTree \( fn \) respectively (Line 7-10). Enumerate all cases of distributing the number of failed services to \( fn \) and \( on \) (for \( fn \), the number of failed services ranges from \( min \) to \( max \)) (Line 11), recursively invoke RelTreeCal and RelForest to calculate subTree probability (the algorithm for RelForestCal is similar and thus is omitted here due to limitation of the space).

- **Line 14:** Add up all subTree probabilities.

### 3.2 General Patterns

In this section, we discuss the tree-based reliability model for the other three composition patterns, namely, parallel, conditional, loop.

**Parallel Pattern**

Reliability of each component web service contributes equally to the overall reliability of the composite web service as in sequential pattern. Therefore, tree-based reliability model for the parallel pattern is the same as that for sequential pattern.

**Loop Pattern**

Assume the maximum number of cycles in a loop is known, then the loop can be represented by a sequence of conditional patterns. Fig. 3 is an example of loop unfolding, where \( p_i \) is the probability that the \((i+1)\)th cycle will be executed after the \( i \)th cycle. Note \( p_k = 1 \). The value of maximum loop and the probabilities can be achieved from the historical record.

![Loop unfolding example](image)

**Conditional Pattern**

Conditional pattern should be treated specially when evaluating the reliability. Each branch has its probability to be executed, and services on the frequent-visited branch have more impact than those on the rarely-visited branch. Therefore, aggregated reliability of conditional pattern could not be simply obtained as in sequential/parallel patterns. The basic idea is to transform the conditional pattern into other patterns, and then apply the above-mentioned approach to calculate its reliability.
Definition 4. Execution Path
A composite web service can have one or multiple execution paths, and all component services in an execution path will be executed in a run, i.e., execution path is free of conditional pattern. For complete definition of execution path, we refer interested readers to [9]. Each execution path is associated with an execution probability.

Fig. 4 is an example of a composite web service and its corresponding execution paths. By default, the split structure is conditional pattern while parallel pattern is annotated with the plus sign. Since the execution path only contains sequential and parallel patterns, we can apply the above the tree-based model and analysis method.

Zheng et al. [10] has proposed a rooted tree-based approach to extract the execution path from composite web service, which could be applied in our context without much modification. For the detailed algorithms, interested readers could refer to the original paper; we just present the basic process as the following:

- **Step 1.** Reduce the parallel and loop patterns to the sequential and conditional patterns as described above. The resulting composition structure is free from parallel and loop patterns.
- **Step 2.** Then the composition structure is transformed into a rooted tree. Each node in the tree represents a component web service, and the root node represents the beginning component web service in the composition. The child nodes represent the component web services that will be executed after the current one. For sequential pattern, the node has only one child node, while for conditional pattern, multiple child nodes exist where each child node represents one branch. Each branch is annotated with the chosen probability.
- **Step 3.** Extract the execution path from the rooted tree. Each path in the rooted tree represents an execution path of the original composite web service. Starting from each leaf node, all execution paths can be identified and their corresponding execution probability can also be calculated.

For each execution path, we can build the RelTree and evaluate its reliability. A comprehensive example is presented in the next section to illustrate the whole process.

4 Application of Reliability Models

4.1 A Running Example

Fig. 5(a) is a composite web service with 14 component services. The parallel pattern is annotated with the plus sign to differentiate from the conditional pattern. Fig. 5(b)
is the corresponding representation for the composite web service after the parallel and loop patterns are reduced. Note that the order of component services in the parallel patterns is not important after transformation. Then we apply the execution path extraction method to generate the rooted tree in Fig. 5(c). In Fig. 5(c) the conditional branches are annotated with the execution probabilities. Note the branches after the node 11 could have different execution probabilities depending on previous executed path. The execution paths can be generated by visiting each leaf node along to the root node. In Fig. 5(c), there are six execution paths for the composite web service, and their execution probabilities are also annotated, e.g. the first execution path (EP1) is (1, 2, 3, 4, 6, 11, 12, 14) with an execution probability of 0.21. By applying the RelTree Building algorithm, we get the RelTree for EP1 in Fig. 5(e) from the Reliability Hierarchy Tree in Fig. 5(d).

![Diagram](image)

**Fig. 5.** Tree-based Reliability Analysis Process

Then the reliability of this execution path can be analyzed using our program developed in Java which is used to facilitate the tree-based reliability process. Table 3 shows the results.
Table 3. Reliability Analysis Result For EP1

<table>
<thead>
<tr>
<th>Descriptions</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability of the component web service</td>
<td>0.9913</td>
</tr>
<tr>
<td>Reliability of EP1 without considering CCF</td>
<td>0.8629</td>
</tr>
<tr>
<td>Reliability of EP1 considering CCF</td>
<td>0.9035</td>
</tr>
<tr>
<td>The probability that at most 2 component web services in EP1 fail simultaneously</td>
<td>0.9935</td>
</tr>
</tbody>
</table>

Fig. 6 shows comparison of reliability values computed using traditional and our CCF-aware approach for each execution path in different scenarios.

- **Scenario 1:** The reliability of each execution path in composite web service, i.e. the probability that no web service fails. (Fig. 6(a))
- **Scenario 2:** The probability that at most one web service failed in the execution path. (Fig. 6(b))
- **Scenario 3:** The probability that at most two web service failed simultaneously in the execution path. (Fig. 6(c))

Also the execution probability is annotated under the corresponding execution path in the figure. The results indicate reliability values calculated by these two methods are different in all scenarios. Our method is superior to the traditional one in that it models the reliability hierarchy and is CCF-aware, which is guaranteed to estimate reliability values for the composite web service more accurately.
4.2 Evaluating a Fault-Tolerant System

The composite web service in Fig. 7 (a) improves its reliability by employing redundant web services. The component services 2 and 3 are invoked concurrently and the success of either one (or both) will be sufficient to continue the next step. If reliabilities of each component service in the composite web service are statistically independent, then the overall reliability can be calculated using the reliability value of each service (as annotated in Fig. 7 (a)), which is

\[
0.9948 \times (1 - (1 - 0.9261) \times (1 - 0.9233)) \times 0.9881 = 0.9774,
\]

thus meets the overall reliability requirement. Unfortunately, in reality, reliabilities of the component services may correlate with each other in certain level.

Fig. 7. Evaluating the Reliability of a Fault-tolerant System

Fig. 7 (b) is one possible RelTree for this composite web service, which indicates that WS2 and WS3 are deployed on the same machine. Thus they will fail simultaneously due to the common-cause failures in host, provider as well as network levels. Using our analysis program, the reliability of the composite web service is only 0.9194, which is far below the threshold. CCFs should be considered when designing the composite web service to get accurate reliability value, especially for those requiring high reliability and using fault-tolerant techniques.

5 Related Work

Reliability is one of the most important QoS metrics for the web service, which has been studied extensively. Tsai et al. [11] proposes a Service-Oriented software Reliability Model (SORM) which deals with both atomic and composite service. For the atomic service, group testing is used to evaluate its reliability efficiently. For the composite service, the overall reliability is evaluated using an architecture-based model. Grassi and Patella [12] focuses on the reliability quality metric for service-oriented applications and presents the RelServ service architecture to support reliability predict for composite service. His method only holds under the hypothesis that component services are independent. Zhong and Qi [6] propose a Petri net based approach to calculate the reliability of web service composition. In his approach, the
web service composition specification is firstly transformed into Stochastic Petri Nets (SPN) model, then standard SPN methods and tools can be applied to analyze the reliability of web service composition. Cardoso et al. [4] models the reliability dimension of tasks in workflows by creating a mapping between reliability block diagrams (RBD) and workflow structures, and he develops the Stochastic workflow reduction (SWR) algorithm to compute QoS metrics (including reliability) for workflows based on task's QoS metrics. Xia et al. [5] proposes a reduction technique to simplify QoS (including reliability metric) of workflow systems based on GWF-net into tasks with qualitatively equivalent QoS metrics. Other structure-based reliability aggregation method for web service composition can be found in [8] [9]. However, all of these approaches do not consider the reliability correlation of the component services and take the assumption of statistically independence between component web services.

Jaeger et al. [13] discusses the dependencies between particular services using dependence domain, and a simple example is used to show that the availability calculated by the micro-perspective approach would be worse than the reality if such dependencies exist. The analysis also applies to the reliability metric, however, they does not present a complete model or QoS aggregation method for web service composition with dependence domain in consideration. To the best of our knowledge, our work is the first to present a model and approach to analyze the reliability of composite web service which considers the reliability correlation between its component web services.

6 Conclusion

In this paper, we propose a reliability model for composite web service considering CCF between component services in host, provider and network levels. RelTree is used to represent the correlation of component services and an analysis method is also presented to calculate the overall reliability. By replacing loop patterns with a sequence of conditional patterns and using execution path extraction technique to reduce the conditional pattern, our modeling and analysis approach are applicable to common composition patterns in the service composition. Our approach can achieve more accurate reliability value for the composite web service, and is especially useful for evaluating fault-tolerant mechanisms.

As the next step, we will research on considering CCF-aware reliability in the process of web service selection for the composition. In addition, we will study other aspects of correlation between component web services.

References


