PERFORMANCE EVALUATION OF SERVICE-ORIENTED APPLICATIONS USING NHPP MODEL AND PETRI NETS

Jiajun Xu¹, Shuzhen Yao¹
¹School of Computer Science and Engineering, Beihang University, Beijing, China
{xujiajun,szyao}@buaa.edu.cn

Abstract

Performance evaluation of a service-oriented application is a critical problem. Markov chain becomes a popular model for performance evaluation. Unfortunately, Markov chains suffer from state-space explosion. Stochastic Petri Nets superior to conventional Markov chain have recently emerged as a promising approach. Methods based on Stochastic Petri Nets currently available face the problem of estimating the unknown failure rate and normally assume that the failure times follow exponential distribution, which could not simulate the cases of other distributions. Therefore, the technique of estimating the failure data and the other probability distribution functions should be considered. This paper applies Non-Homogeneous Poisson Process (NHPP) model to estimate the failure data; a Non-Markovian Stochastic Time Petri Net (NMSTPN) is defined and an algorithm of constructing the reachability tree is presented. Then, a procedure of computing the probabilities of the sequences of failures in which the firing rates are non-Markovian with (but not limited to) exponential is presented. The whole approach is validated through an example. As an extension of the conventional methods, the resulting framework can act as a useful systematic approach in performance evaluation of complex applications. This approach is helpful to managers in analyzing the overall quality of the service-oriented application.

Keywords: Performance evaluation, service-oriented application, NHPP, Petri Net

1. INTRODUCTION

Service-oriented architecture is a favored framework for building distributed and heterogeneous applications by orchestrating loosely coupled primitive or composite services [1]. The compositional characterization of multiple services and the unpredictable behavior of single component pose a new challenge for the performance assessment of service-oriented applications [2]. As the complexity of service-oriented systems increases, performance evaluation is turning into a harder task because the performance of the systems depends not only on all system components failing/being in the failed states, but also on the sequences of occurrences of those failures.

Performance evaluation of service-oriented applications that takes the failures into consideration provides a realistic view of system failure analysis. The conventional black-box approach ignores the structure information of applications and thus becomes unsuitable to model these service-based systems. As popular white-box approaches, the architecture-based performance models are brought up to describe the inner behaviors of these systems. The state-based approach is the major technique among architecture-based performance models and assumes that the transfer of control between components has a Markov property, uses Markov chain to describe the software structure and calculate the software performance. These Markov performance models can further be divided into discrete time Markov chain (DTMC), continuous time Markov chain (CTMC) or semi Markov process (SMC). Unfortunately, Markov chains suffer from state-space explosion.

Stochastic Petri Nets (SPNs) superior to conventional Markov chain have recently emerged as a promising approach for modelling service-oriented applications [3], [4], [5], [6]. The major advantages of SPNs over Markov chains include: SPNs can represent many states in a concise manner; SPNs capture precedence relations and structural interactions; SPNs can model deadlocks, conflicts, and buffer sizes; SPNs can model multi-resource constraints; SPNs have an underlying mathematical foundation that can be exploited to perform qualitative and quantitative analysis of the system; and since SPNs as graphical tools are derived from the logical structure of the application, they are easy to understand and communicate to others. Usually, in these SPNs-based approaches no information other than the failure data is used. However, on the one hand, the current SPNs-based methods are facing the problem of estimating the unknown failure rate. To obtain the complete failure rate, it is required to wait until the end of the service life cycle, but often requires a long time to obtain the complete information. In most circumstances, this kind of time-consuming method is difficult to use. On the other hand, conventional SPNs-based approaches usually assume that the failure times follow the exponential distribution, which could not simulate the cases of other distributions. Therefore, there need to be more work on the estimation of the failure data and the cases of other probability distributions to evaluate the overall quality of a service-oriented application.

This study aims to develop a convenient framework for designers to analyze the sequential failures for improving the overall performance of the service-oriented applications. The distinguishing feature of our approach lies in adopting NHPP model to estimate the failure data, utilizing Non-
Markovian Stochastic Time Petri Net (NMSTPN) for capturing the non-Markovian behavior of the applications, exploring potential failure sequences due to the combination of unexpected failures. The probabilities of the sequences of the failures are computed with an approximation method in which the firing rates may obey a non-exponential distribution. Example of a service-oriented system demonstrates its usability and effectiveness. The resulting framework extends the traditional methods and can act as a systematic approach for evaluating the performance of complex applications. The results obtained can contribute to the performance aspect in designing and implementing service-oriented applications.

The remainder of the paper is organized as follows. In Section 2, the related work is presented. In Section 3, we introduce the modeling of service composition using NMSTPN. In Section 4, we present the performance evaluation framework and the mathematical equation of the methodology. Section 5 validates the approach through an example. Section 6 concludes the paper.

2. RELATED WORK

For performance evaluation, a number of analytical models have been proposed to address the problem of quantifying the software performance. Goseva-Popstojanova et al. [7] consider the problems that the complexity of system is high and the number of components is large and thus develop the performance models: operational profile-based approach, state-based approach, and path-based approach. The operational profile-based approach uses user frequency information to assess performance. The state-based approach assumes the transfer of control between modules has a Markov property, uses Markov chain to describe the software structure and calculate the software performance. The path-based approach explores all possible execution paths and then calculates the reliability of the entire software. These approaches integrate the performance of each component and then compute the system performance according to the system structure information.

The performance models for sequential failure analysis have been developed to determine the system performance requirements. One prominent tool, called Fault Tree Analysis (FTA) method and proposed in the early 1960s, only statically represents the various combinations of possible events which makes it difficult in handling complex systems. A variety of failure analysis techniques, including Failure Mode and Effect Analysis (FMEA), share a similar deficiency [8], [9]. There has been a substantial research on using Sequential Failure Logic (SFL) to capture the occurrence of ordered failures. Failures in the components can cause the failure of the whole system. In Fussell’s research on SFL [10], the focus is on evaluating non-repairable systems and exact/approximate techniques are applied to compute the likelihood of appearance of the elementary events which are assumed to be independent and static [11]. It is valuable, but the applications of this method are fairly limited, because the available failure analysis methods always use ordered sets of failures to evaluate the likelihood of their occurrence, which cannot simulate real life cases; therefore, the sequential failures and the likelihood of their occurrence should be explored. Obviously, there is a need for more work in this field.

The behavior of interconnected components of a complex system and their unexpected interactions of failures cannot be suitably identified by the conventional techniques. As the increasing structure and dynamic behaviors of the components, Markov models are applied to solve these problems. These models can be divided into DTMC, CTMC or SMC [12]. As one of the most representative methods in performance evaluation, Cheung et al. [13] use DTMC to describe the control flow of components, encode the configuration information into the model to calculate the failure probability. Most subsequent Markov methods are based on Cheung method, such as Goseva-Popstojanova et al. [14] use UML diagrams with reliability data to model system structure and DTMC to evaluate reliability, Sato et al. [15] use Markov model to describe control flow structure of service composition, Ren et al. [16] use the service dependency graph to model service composition and the DTMC method to calculate the performance. However, with the increase of the number of components, the models face the state space explosion problem. Dugan et al. [17] have shown through a process known as modularization, to apply different Markov models on each identified independent sub-tree. Although it is useful in certain fault tolerant systems, it causes state space explosion problem due to the increasing quantity of components. Rao et al. [18] propose a numerical integration technique to reduce the problem of state-space and minimize the computational time. Also, maintaining the significance of sophisticated modeling to detect failures in dynamic systems, some work contributed significantly to the development and application of the approach [19]. However, state space of Markov models becomes too large to calculate as the quantity of components grows. Developing new methodology to identify the failure sequences is essential.

Petri Net technique is superior to conventional Markov chain techniques since the size of its model grows slightly with increasing complexity of the system [20]. The Petri Nets with exponentially distributed random time delays are referred to as SPNs [3] and have been employed to assess the performance of the services [6]. The major advantages of Petri Nets over Markov chains include: Petri Nets can represent many states in a concise manner; Petri Nets capture precedence relations and structural interactions; Petri Nets can model deadlocks, conflicts, and butler sizes; Petri Nets can model multi-resource constraints; Petri Nets have an underlying mathematical foundation that can be exploited to perform qualitative and quantitative analysis of the system; and since Petri Nets are derived from the logical sequencing of the system and are graphical, they are easy to
understand trends in performance evaluation of service composition using Petri Nets. The composite service is modeled and analyzed through transforming the basic and structured activities of service composition to Petri Nets [5]. For example, Zhao et al. [21] use SPNs to detect service failures and have solved a real world issue. Perez et al. [22] develop an approach to model the self-adaptive behavior of service-oriented software systems with SPNs, to analyze and improve their performance. Wang et al. [4] use SPNs to assess the performance of systems with the help of Markov isomorphism. General-purpose tools for solving SPN models include GreatSPN [23], SPNP [24], and PIPE [25]. Most literature is on SPNs where transition firing times are mutually independent exponentially distributed random variable. However, transition firing times might not be exponentially distributed, for instance, Weibull distribution has been a lively topic of debate for performance evaluation [26], [27], [28] so that the current approach is inadequate. Moreover, in reality, little current research effort is being dedicated to the analysis of the explicit sequential failures of service-oriented applications. Besides, most existing methods lack failure data, because in order to get complete failure information, it is needed to wait until the termination of the service life cycle. But service normally runs with high reliability and small sample, thus it often requires too long to get the complete failure data. Both problems in a certain extent restrict the usability and effectiveness of the current methods in performance evaluation. Therefore, this paper proposes the methods of estimating the failure rate data and the technique of evaluating the performance with non-exponential distributed failure times.

3. Modeling of Service Composition

This section presents NMSTPN models of service composition. Firstly, we introduce the basic theory of NMSTPN, in which its firing time obeys general distribution. Next, we will show the basic structure model of service composition based on NMSTPN. Finally, we present the failure rate evaluation method and NMSTPN model of service composition with failure information.

3.1 NMSTPN Theory

First invented by Carl Petri in the early 1960s, Petri Net is widely used for simulating the local behavior and combination characteristics, to analyze the performance of most of complex system [29]. Timed Petri Net extends Petri Net with a time delay on each transition. Generalized Stochastic Petri Net is the expansion of Timed Petri Net by introducing intermediate transitions and exponentially timed transitions. Deterministic and Stochastic Petri Net extends Generalized Stochastic Petri Net with deterministically timed transitions. These Petri Nets allow the stochastic delay to be exponential.

NMSTPN extends SPNs that assumes the delay to be deterministic or exponential. Objectively, there exist such applications, in which the duration of their events follows a continuous non-negative exponential distribution. The theoretical limitations in the traditional Petri Nets should be relaxed for qualitative analysis. NMSTPN allows the delay on the transitions is not restricted to general distribution and thus has wider application range.

Definition 3.1 (Non-Markovian Stochastic Time Petri Net (NMSTPN)). NMSTPN is a directed bipartite graph defined by a 7-tuple $N = \{P; T; A; M_0; I(t); O(t); \lambda\}$

1) $P = \{p_1, p_2, \ldots, p_n\}$ is a set of places, where a place is used to represent the state.

2) $T = \{t_1, t_2, \ldots, t_n\}$ is a set of transitions, where each transition representing an event or an action.

3) $A \subseteq \{T \times P\} \cup \{P \times T\}$ is a set of directed arcs. $I(t) = \{p| (p, t) \in A\}$ is a set of input places of a transition $t$. $O(t) = \{p| (t, p) \in A\}$ is the set of output places of a transition $t$.

4) $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ ($i = 1, 2, \ldots, n$) is the firing rate set on the transitions. The firing rates are not limited to exponential.

5) $M_0$ is the initial state (the marking). Each place can contain tokens and the present marking is represented by the number of tokens within the place.

A service oriented application behavior is basically a consequence of events or actions. These events or actions can be represented by the firings of transitions and the states of the service are represented by places. To be more specific, the behavior of a NMSTPN model follows two firing restrictions: (1) enable rule: there are tokens in all the input places of a transition suggest that the firing conditions have been satisfied, and (2) firing rule: when firing, a transition removes one token from each of its input places and deposits one token in each of its output places. The shift of tokens in the places simulates the behavior and thus enables analyzing the potential failures.

Definition 3.2 (Pre- and post-conditions). A transition $t_i$ has a certain number of input (output) places, generally names as pre-conditions (post-conditions).

1) $t_i := p_j \in I(t_i)$. is the set of the pre-conditions of transition $t_i$, i.e., the set of all places have an outgoing edge to $t_i$.

2) $t_i := p_j \in O(t_i)$. is the set of the post-conditions of transition $t_i$, i.e., the set of all places have an outgoing edge to $t_i$.

Pre- and post-conditions are defined analogously to pre- and post-conditions.

According to the above theorem, the primary use of NMSTPN is the representation of constructing the relationship between cause/effect and providing a generally formal method to capture all potential states for analysis. Applying NMSTPN into system performance evaluation enables designers to analyze the quality factors due to a combination of undesired failures and the sequences. NMSTPN enables analyzing the combined failure models.
not only to forecast the potential severity, but also to investigate the probability of appearance of failure models, so that designers can develop proper protection mechanisms to avoid the negative impacts of these failures.

### 3.2 Basic Structure Models of Service Composition

This section studies the basic structure model of service composition based on NMSTPN. The basic structure model is constructed based on the structure information of service composition.

<table>
<thead>
<tr>
<th>Structures</th>
<th>NMSTPN models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td><img src="sequence.png" alt="Sequence Diagram" /></td>
</tr>
<tr>
<td>Switch</td>
<td><img src="switch.png" alt="Switch Diagram" /></td>
</tr>
<tr>
<td>Flow</td>
<td><img src="flow.png" alt="Flow Diagram" /></td>
</tr>
<tr>
<td>While</td>
<td><img src="while.png" alt="While Diagram" /></td>
</tr>
<tr>
<td>RepeatUntil</td>
<td><img src="repeatuntil.png" alt="RepeatUntil Diagram" /></td>
</tr>
</tbody>
</table>

Table I. NMSTPN models of service composition.

Using NMSTPN to model service composition is the first step to evaluate the performance of service-oriented application. Service composition generates more complex and more powerful services and is described through different main logical structures, as illustrated in table I. *Sequence* structure defines the running order and the serialization of single services. *Switch* structure defines the if/choice condition. *Flow* structure defines the concurrent execution of services. *While* and *repeatUntil* structure defines the loop situation. Different service interaction reflects different cooperation intention, thus forms different structure. In theory, NMSTPN can model system behavior patterns such as concurrent, competition, conflict, and synchronization, and verify a series of properties that are closely related to the software performance. NMSTPN model shows a relationship between the firing time and an arbitrary distribution function of the trigger probability, to form a basis of performance evaluation. Service composition can be modeled by transforming its structures to different NMSTPN models [5] and NMSTPN models of the major structures are listed in table I.

![Figure 1](figure1.png)

**Figure 1. The rule for transforming the pick structure to a basic structure**

In addition to the above basic structures, there are some more complex structures, which cannot be directly modeled using the above method. However, a complex structure can be transformed into a basic structure, that is, the combination of several basic structures is used to represent the complex structure, so that the complex NMSTPN model is transformed into an equivalent simple one [30]. As shown in Fig. 1, the rule for transforming the non-basic structure *pick* to the basic structure is given.

### 3.3 Estimation of Failure Rate using NHPP Model

Firstly, this section discusses the failure behavior of a single service using NMSTPN. The behavior of a basic service can be modeled as a NMSTPN model, as depicted in Fig. 2. After a service is invoked, there are two optional states: the normal executed state and the failure state. For the normal executions of services, the corresponding rates are 1s. For the failure behavior of service, their failure rates are unknown. In other words, the failure rate of a single service is unknown.

![Figure 2](figure2.png)

**Figure 2. NMSTPN model of a basic service**

To solve the problem that the failure rate of a single service is unknown, the failure rate evaluation method based on Non-Homogeneous Poisson Process (NHPP) is adopted
in this section. In the past forty years, various Software Reliability Growth Models (SRGMs) evaluate system performance using either overall system failure time or the number of faults at a specific time [31], [32], [33], [34] and generally agree that the fault detection process is a NHPP. NHPP model is based on key assumptions of software development and models the software failure process according to the relationship between the number of detected faults and testing time, to predict actual software failure information, including the failure rate. A surging number of service-oriented software systems have been developed to support interoperable machine-to-machine interaction over the network and NHPP model is adequate for these systems [35], [36].

NHPP model is used for calculating failure rate of each web services but with diverging assumptions of the fault detection rate (FDR). Goel et al. [37] expanded earlier works of Jelinsky and Moranda [38] and derived the classic SRGM with constant FDR, commonly known as the Goel-Okumoto (G-O) model. A group of researchers [39], [40] later asserted that constant FDR cannot accurately simulate the dynamical process of fault detection; instead they proposed a time-dependant FDR to reflect its smooth/regular change in time. SRGMs in such category include: S-shaped model [41], SRGM with a test-effort function [42], and the inflection S-shaped model with a change-point function [43]. To sum up, most models assume a deterministic relationship between the cumulative number of errors detected by software testing (or the time-interval between software failures) and the time span of the software fault detection. However, the aforementioned assumptions of deterministic FDR would no longer hold as environmental noise introduces great uncertainty for software development in the testing and operational phase. Characterized with decentralized organizations, individual autonomy and interactive openness, the network environment introduces great uncertainties and hence significantly affects the traditional detection process model. To address such challenges, new stochastic models [44], [45], [46], [47] were proposed, which assume random and independent failure occurrences during the fault detection process. Those models generally treat the environmental factors collectively as a standard Gaussian white noise.

In this paper we adopt the continuous-time NHPP [45], [46], [48], [49], to model the fault counting process \( \{N(t), t \geq 0\} \). Widely used in many practical fields [42], NHPP model assumes the system (1) is subject to failures at random times caused by the remaining errors in the system; (2) the mean number of faults detected in the time interval \((t, t + \delta t)\) is proportional to the mean number of remaining faults. If we define the random variable, \(N(t)\), whose mean value function is known as \(M(t)\), then an SRGM based on NHPP can be formulated as a Poisson process:

\[
P[N(t) = n] = \frac{(M(t))^n e^{-M(t)}}{n!} , n = 0,1,2,...
\]  

Furthermore, \(M(t)\) is formulated as \(\dot{\lambda} = dM/dt\), where \(\dot{\lambda}\) is the failure rate function. If the number of faults detected by the current testing process is proportional to the number of remaining faults, then in a continuous state space, the mathematical expression of NHPP model is routinely described as

\[
\frac{dM}{dt} = b(t)(a - M)
\]  

The behavior of \(b(t)\) is not completely known since it is subject to random effects such as the testing effort expenditure, the skill level of the testers, and the environment and thus might have irregular fluctuation. Thus, since \(b(t)\) is subject to a number of random environmental effects [50], we follow previous studies of Yamada et al. [45], [47] and represent \(b(t)\) as a sum of its mean value \(\langle r(t) \rangle\) and a zero-mean (random) fluctuation \(\sigma r(t)\):

\[
b(t) = \langle r(t) \rangle + \sigma r(t),\quad \langle r(t) \rangle = 0
\]  

We adopt a simple expression for the mean fault-detection rate per remaining fault from the generalized G-O model [51]

\[
\langle r(t) \rangle = d\dot{r}
\]  

where \(d\) is the shape factor and \(b \in (0,1)\) is a constant representing the fault detection rate per fault, in other words, when \(b = 0\) shows a deteriorating debugging performance.

By substituting (3) and (4) into (2), we obtain a stochastic equation:

\[
\frac{dM}{dt} = \left[b\dot{r} + \sigma r(t)\right](a - M)
\]  

where \(b(t)\) is the fault detect rate function, \(b\) is the initial fault detect rate, \(a\) is the initial number of faults, \(d\) is the shape factor, and \(\sigma r(t)\) is the Gaussian noise.

After solving the above differential equation (5), the failure rate function is exhibited as follows:

\[
\dot{\lambda}(t) = a(b\dot{r} - \frac{\sigma^2}{2}) \exp(-\frac{b\dot{r} + \sigma^2}{d + 2\sigma^2})
\]  

There are four input parameters in our failure rate function, including the total number of initial faults \(a\), the fault detection rate \(b\), the shape factor \(d\), and the noise strength \(\sigma\). We employ earlier white-noise model [47], [49] to estimate the correlation strength \(\sigma\). Given a data set, the methods including method of moments, least squares method, maximum likelihood and Bayesian methods, can be used to estimate three parameters: \(a\), \(b\), and \(d\).

3.4 NMSTPN Model of Service Composition

This section presents the procedure of constructing NMSTPN model of service composition with failure information, as depicted in Fig. 3. NMSTPN model is constructed on the basis of the essential structure of service composition and the failure information of a single service.

Given a business model, the basic structure model based on NMSTPN is built according to the logical structure of the service composition, as shown in table I. Through the service composition model, one can build NMSTPN model based on basic structure model. The basic structure model of
service composition considers the normal state of the service execution. However, in order to analyze the failure behavior of all services in service composition, it is necessary to model the failure behavior of each single service. Therefore, based on the basic structure model, each service is integrated with its failure information (failure rate), so that NMSTPN model with failure states is the target model.

![Basic structure of service composition using NMSTPN](image)

**Figure 3. Procedure of constructing NMSTPN model of service composition**

In order to demonstrate the construction process of NMSTPN model of service composition, we take two concurrent services (Service1 and Service2) as an example, as shown in Fig.4. Fig.4 (not containing the dashed box) shows the basic structure model of two concurrent services using NMSTPN. On the basis of this essential structure model, the two services are introduced with their failure states (state Failed 1 and state Failed 2), as shown in the dashed box of Fig.4. Finally, Fig.4 (including the dashed box) is the target NMSTPN model of two concurrent services with failure information.

![NMSTPN model of two concurrent services](image)

**Figure 4. NMSTPN model of two concurrent services**

### 4. Performance Evaluation Framework

In this section, we present the framework to estimate the performance of the underlying system. The framework generates the reachability tree of NMSTPN model, incorporates the reachability tree for exploring all sequential failures and includes a computation procedure to calculate the probability of occurrence of each sequential failure.

#### 4.1 Construction of Reachability Tree

Many application problems are concerned about the possible state of the system, and the reachable state set is the key to solve this problem. Reachable state space is a collection of all the possible states of NMSTPN model.

A NMSTPN with a specific initial marking is denoted by \((N, M_0)\). A token is defined as a message and an arc indicates the passing mechanism of the message. Each transition means a message processing unit. The marking (or also called state) representing the number of tokens within each place, is an \(n \times 1\) column vector \(M\) with non-negative integer entries: \(M = [m_1, m_2, \ldots, m_n]^T\), where \(m_p\) is the number of tokens in place \(p\). As defined above, \(M_0\) is the initial marking of the system. All the reachable markings from \(M_0\) are gathered to generate the state space of NMSTPN model. The incident matrix \(A\) of NMSTPN \((N, M_0)\) with \(m\) places and \(n\) transitions is an \(m \times n\) matrix of integers:

\[
A = \begin{bmatrix} a_{ij} \end{bmatrix}, \text{ where } a_{ij} = a_{ij}^+ - a_{ij}^-
\]

(7)

where \(a_{ij}\) is result of the post-conditions of transition \(j\), minus its pre-conditions. The state equation as the mathematical representation of generating the state space is denoted by:

\[
M_{k+1} = M_k - A^T U_k
\]

(8)

where \(M_k\) is the marking after \(k\)th firing, specified as an \(n \times 1\) vector of nonnegative integers. The \(k\)th control vector \(U_k\) is an \(m \times 1\) vector with a 1 in the \(i\)th position and zeros elsewhere, clarifying the transition \(i\) fires in the \(k\)th execution.

**Definition 4.1 (Reachable state space).** The reachable state space \([M_0^+]\) of NMSTPN \((N, M_0)\) is a set satisfying the following conditions:

1) \(M_0 \in [M_0^+]\);

2) If there exist \(M' \in [M_0^+]\), \(T \in T\), leading to \(M' \land T > M\), then \(M \in [M_0^+]\).

Therefore, one can get: If \(M \in [M_0^+]\), there is a sequence \(\sigma = M_0 T_1 M_1 T_2 M_2 \cdots T_n M_n\), and \(\forall k, 0 < k \leq n\), \(M_{k-1} \land T_k > M_k\), and \(M_n = M\). And vice versa.

**Definition 4.2 (Reachability tree).** The reachability tree of NMSTPN \((N, M_0)\) is a tree whose node is a marking in reachable state space and arc is marked by a transition.

Algorithm 1 presents the process of constructing the reachability tree. The algorithm starts from the initial state \(M_0\). Suppose \(M\) is the current marking and under \(M\), a set of transitions are fireable. Any enabled transition in the marking \(M\) may fire, resulting in a new marking \(M'\). The reachability tree can be constructed from \(M\). If \(M'\) equals to a marking in the reachability tree, one can draw an arc from \(M\) to \(M'\). If \(M'\) is larger (or smaller) than the current marking in the reachability tree, one can increase (or decrease) the weight of the related arc. This procedure continues until there is a final marking under which no transition can be fired. A duplicated node is a node already in the tree. Method node(M) represents a node corresponding to the state \(M\) in the tree. Method create(M) represents creating a node corresponding to the marking \(M\). \(M_1 \rightarrow M_2\) represents that \(M_2\) comes from \(M_1\) when transition \(T\) fires and \(T\) means an arc from \(M_1\) to \(M_2\).
Algorithm 1 Constructing the reachability tree of NMSTPN \((N, M_0)\)

**Input:** A NMSTPN \((N, M_0)\)

**Output:** Reachability tree

1: Initialize the marking of NMSTPN as \(M_0\), create(M0);
2: Initialize reachable state space \(M_{set} = \emptyset\);
3: Create Persistent Set \(H\);
4: Set the current node to node(M);
5: for each \(T_i \in M\) \(\exists T_i \supseteq M\) do
6: \(M[T_i] > M\);
7: if \(H\) is not empty then then
8: \(M_{set} + = H\);
9: end if
10: if \(M == \emptyset\) then
11: create(M);
12: \(node(M) \rightarrow T_i \rightarrow node(M')\);
13: \(M_{set} = M'\);
14: else
15: for each \(M_i \in M_{set}\) do
16: if \(M' = M_i\) then
17: \(node(M) \rightarrow T_i \rightarrow node(M_i)\);
18: end if
19: if \(M' = M_0\) then
20: \(node(M) \rightarrow T_i \rightarrow node(M_0)\);
21: else if \(M' > M_i\) then
22: Increase the weight of the related arc;
23: else if \(M' < M_i\) then
24: Decrease the weight of the related arc;
25: else
26: create(M');
27: \(node(M) \rightarrow T_i \rightarrow node(M')\);
28: \(M_{set} = M_{set} \cup M'\);
29: end if
30: end for
31: end if
32: end for
33: return Reachability tree;

Figure 5 describes the reachability tree of NMSTPN model for two concurrent services. \(M_0, M_1, M_2, M_3, M_4\) and \(M_5\) stand for the normal states of service composition, while \(M_4\) and \(M_6\) stand for the failure states.

**Output:** The probability of system failures

1: Calculate the failure rates of each service, due to failure rate function derived from NHPP model;
2: Explore the failure sequence set \(E\), according to the reachability tree;
3: for Each failure sequence in \(E\) do
4: Find all the transitions \(T = \{t_1, t_2, \cdots, t_m\}\) leading to the system failure;
5: Find all the failure rate \(\lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_m\}\) of the transitions;
6: for Each transition in one failure sequence do
7: Calculate the weight of transition \(j, j \in 1, \cdots, m\): 
\[
\omega_j = \sum_{i=1}^{m} \lambda_i
\]
8: Calculate the waiting time \(t_j\) of transition \(j\) with the total time \(T\):
\[
t_j = T \cdot \omega_j;
\]
9: Calculate the probability of the transition \(j\) from the probability function.
10: Calculate the probability of each failure sequence.
11: end for
12: Calculate the probability that the system will fail by summarizing all the failure sequences.
13: end for
14: return The probability of system failures;

4.2 Performance Evaluation using Reachability Tree

Algorithm 2 demonstrates the performance evaluation framework. The algorithm takes the reachability tree of NMSTPN model as input for exploring all sequential failures and returns the probability that the application would fail in a total time \(T\). According to the reachability tree, the sequences of failures are recorded through tracing the paths from the initial marking towards the failure markings. By tracking the paths that result in system failure markings, the transition sequences of failures can be obtained. For each identified failure sequence, a computation procedure is presented to calculate the probability of occurrence of that sequence. The probability that the application will fail can be calculated. Finally, the probability that the system will not fail is \(1 - F\).

![Figure 5. Reachability tree of NMSTPN model in Fig. 4](image-url)
4). $Q_{M_1,M_2}(t)$ the transitional probability, i.e. the probability that marking $M_1$ moves to $M_2$, as the transition $t_j$ is firing in some time no greater than $t$.

5). $F_j$ probability of changing from the starting marking $i$ to the ending marking $j$ after some sequential transitions.

As mentioned earlier, the reachability tree represents all (reachable) markings of the system and shows all possible firings at each marking. Hence, based on the failure sequences observed from the reachability tree, the probability of occurrences of a specific sequential failure can be calculated.

In contrast to SPNs whose random delays are exponentially distributed and which are equivalent to Markov chain, in the current approach, the firing times of the transitions obey continuous probability distribution, such as Weibull, Exponential, Normal, Lognormal, Gamma, etc. Formally, if the firing rate $\lambda(x)$ is a random variable with probability density function $f(x)$, its probability of falling into an interval $[a, b]$ can be denoted by the integral:

$$F(x) = \int_a^b f(x)dx$$  \hspace{1cm} (9)

We use the firing times following Weibull distribution as an example to demonstrate the procedure of computing the transitional probability of the sequential failure. Thus, we can write the probability function for transition $i$ as:

$$F_i(t) = \int_0^t \lambda(x) e^{-\lambda(x)t} dx$$  \hspace{1cm} (10)

In a reachability tree, we can consider three possible branching cases and give three lemmas of computing the transitional probabilities:

**Lemma 4.3.** There are no branches. Since the firing time for the transition $t_k$ is not greater than $\tau$, the transitional probability of changing from the marking $M_i$ to $M_j$ in general case can be written as [52]:

$$Q_{M_i,M_j} = F_i(t) = \int_0^t \lambda(x) e^{-\lambda(x)t} dx$$  \hspace{1cm} (11)

**Lemma 4.4.** There is only one branch. In that case, when there is an uncertainty, as to which transition should be fired we have to exercise a different approach for computing the probability of firing a certain transition. Therefore, we have to estimate the probability of the transition $t_k$ to be fired, while the alternative transition $t_l$ is not fired until time $\tau$. The probability that transition $t_k$ is not fired until time $\tau$ is given by $1-F_i$. Then we can write the following [52]:

$$Q_{M_i,M_j} = \int F_d F_i(t) = \int \lambda(x) e^{-\lambda(x)t} dx$$  \hspace{1cm} (12)

**Lemma 4.5.** There is more than one branch. This is a general case where the number of branches is greater than one. Therefore, we have more than one transition to fire. In this case, we should estimate the probability of the transition $t_k$ to be fired, while the alternative transitions $t_0, t_1, \cdots, t_n$ are not fired until time $\tau$. Then the transitional probability $Q_{M_i,M_j}$ of changing from the marking $M_i$ to $M_j$ can be [53]:

$$Q_{M_i,M_j} = \int \lambda(x) e^{-\lambda(x)t} dx$$  \hspace{1cm} (13)

It is worth mentioning that when $\beta = 1$, the probability function becomes the exponential probability density function. Based on these three possible cases, we can obtain the transitional probabilities of the all possible changes of markings. Moreover, $Q_{M_i,M_j} = 0$ if $M_i = M_j$, which means that if the marking does not change, the transitional probability is 0.

To calculate the probability of appearance of sequential failures, a stochastic process is introduced. A stochastic process is a collection of random variables $\{X(\Phi) : \Phi \in \mathcal{A}\}$, defined over the interpretation of probability, with a varying parameter $\phi$ over a set of indexed parameters $\mathcal{A}$. It involves a sequence of random variables called states and the time series associated with these random variables. State space, defined as: $X = \{X_1, X_2, \cdots, X_n\}$, can be continuous [54].

According to the probability theory with stochastic processes, a probability $F_i$ of all states $i, j \in E$ can be denoted by:

$$F_i = P\{X = j, X_1 = i, X_2, \cdots, X_n = i\}$$  \hspace{1cm} (14)

We can consider a process of any applications that is observed at discrete times and the recorded marking is simultaneously one from the markings $M_i$ denoted as $M_1, M_2, \cdots, M_n$. After recording the present marking, a transition must be chosen to be fired and we define $T$ as finite to represent the sequence of all transitions.

If the marking at the time $n$ is $M_i$ and a transition $t$ has been selected, the subsequent marking can be identified with regard to the transition probabilities $Q_{M_i,M_j}$.

If the present marking at time $n$ is denoted as $M_n$, then the above formula equals the mean that:

$$P\{X_n = j, M_n, T_n, \cdots, M_s = i, T_s = t\} = Q_{M_i,M_j}$$  \hspace{1cm} (15)

Thus, the transition probabilities depend upon the present state and the following firing transition.

**Proposition 4.6.** For a given sequence of events $E$, the probability $F_i$ is calculated as:

$$F_i = \prod_{i,j \in E} Q_{M_i,M_j}, \text{where } i, j \in E$$  \hspace{1cm} (16)

**Proof.** For a process, $V = \{V_1, V_2, \cdots, V_n\}$ is a vector meaning the firing order of transitions, where $V_i = 0$ if the transition $i$ is not fired and $V_i = 1$ if the transition $i$ is fired.

Let us introduce a structural function $\Phi(V)$. When the state vector is $V$, $\Phi(V) = 0$ if the sequence of transition is not fired, while $\Phi(V) = 1$ if the sequence of transition is fired.

A sequence of transitions is the chain of events for the linearization of an occurrence of transition. Thus, it can be fired on condition that all transitions in it can be fired. Thus, $\Phi(V)$ is considered as 1 when $V_1 = V_2 = \cdots = V_n = 1$, and 0 otherwise, therefore:

$$\Phi(V) = \min\{V_1, V_2, \cdots, V_n\} = \prod_{i=1}^n V_i$$  \hspace{1cm} (17)

$F_i$ is the probability that the process starts from initial state $i$ and will end in failed state $j$. The transitions fired are independent of each other. Thus, $F_i$ can be calculated from the product of the probability that each transition is fired in the sequence.

$$F_i = P[\Phi(V) = 1] = \prod_{i=1}^n p_i$$  \hspace{1cm} (18)

where $p_i$ is the probability that transition $t_i$ is fired.
Then the probability $F$ that the application will fail can be determined by:

$$F = \sum_{i} (E_{ij})$$

where $h$ means the number of potential sequences from initial state $i$ to final state $j$.

### 4.3 Complexity of the Evaluation Algorithms

To analyze the complexity of two algorithms, it is assumed that NMSTPN has $m$ transitions and $n$ places.

Now we investigate algorithm 1 and algorithm 2. For the algorithm 1, efficiency plays an important role in constructing the reachability tree. For each service runs concurrently, their semantic interactions can be described as a hypercube. The running of each service is independent and all running paths reach the same end node. Because the model has $n$ components, the reachability tree in the state space becomes a hypercube and the hypercube contains $(m-1)^n$ nodes and $n^*(m-1)^n$ edge (arc). For algorithm 2, suppose the model has $r$ failure path, the largest number of transitions is $m$ ($r$ and $m$ are constant), its complexity is linear. Therefore, we can clearly see the complexity of the evaluation algorithms is determined by the algorithm 1 and is generally exponential.

Obviously, when the application contains several concurrent services, the state space of NMSTPN model will increase exponentially as the number of the concurrent services increases. Therefore, when the number of concurrent services is large, it is impossible to directly construct the reachability tree, leading to the problem of state space explosion. However, when we further study NMSTPN model and their corresponding reachability tree, we find that there are some redundant states in the reachability tree. These redundant states can be reduced to reduce the size of the reachability tree, so that the reduced state space is equivalent to the origin state space. The common practice is to reduce and compress the state space is partial order reduction. The main goal of partial order reduction is to solve the state space explosion problem because of the asynchronous running of the concurrent services. The main idea is based on the fact that a system is composed of concurrent services and the executions of different services will produce different order. The execution of each concurrent service can be regarded as a linear sequence of the continuous insertion of each event. $n$ services can generate $n$ events, to create $n!$ linear sequence. In these sequences, some events may have nothing to do with the running order. Therefore, if these events are fixed to the positions in the sequence in advance, one can avoid redundant paths and alleviate state space explosion problem.

Taking Fig. 5 as an example, one can see state set $M = \{M_1, M_2, M_3, M_5\}$. $M_1$ is dependent on $M_2$ and $M_5$, $M_2$ is dependent on $M_3$ and $M_5$, and $M_2$ and $M_3$ are independent from each other. State sequence $M_1M_5M_2M_5$ has the same meaning as $\{M_1M_2M_5M_5, M_1M_2M_3M_5\}$. Because $M_1$ and $M_2$ are neighbors and mutually independent, after swapping their positions, one can get a new sequence $M_1M_1M_2M_5$. It is clearly that $M_3$ and $M_5$ are fixed positions, which constitute a persistent set $\{M_1, M_3\}$. When the algorithm 1 comes to the persistent set, it just visits the persistent set, so that it can effectively reduce the number of steps in constructing the reachability tree.

### 5. Evaluation and Discussion

The evaluation is divided to four parts: modeling of the underlying system, estimation of failure rates, a qualitative performance assessment, and a sensitivity analysis.

#### 5.1 NMSTPN Model of an Illustrating Example

To assess and optimize the performance of the application, the behavior of the service composition is modeled by NMSTPNs. NMSTPNs are grounded on a solid mathematical foundation combined to an intuitive high-level user interface. This allows constructing formal, yet simple, descriptions of systems characterized by sequence, switch, while, parallel (flow), etc., which appear quite often in the service-oriented application. The service is a port based component and is equivalent to the sub-net of NMSTPN model. The service composition model could be constructed from a simple service model and the increasing size of

![Figure 6. The travel plan (TP) composite service](image-url)
service composition can enlarge the unpredictability of the sequential failures. Thus, we use a representative travel plan (TP) composite service to demonstrate the method. As depicted in Fig. 6, the Bike Reservation and Car Reservation

Figure 7. NMSTPN model of the TP service

Figure 8. Reachability tree of NMSTPN model depicted in Fig 7
services are in a choice pattern labeled with $\alpha$ for one path showing the probability of choosing that path. Then, two services integrated with Map Search, and Flight Reservation integrated with Hotel Reservation service are both in sequence. Then, the sequence services (Flight Reservation and Hotel Reservation) are in flow (parallel) with Attraction Search, to construct a loop pattern labeled with $\beta$ counting the number of iterations.

In this process, the enterprise can implement a service level agreement (SLA) that states that performance will be provided within 200 business hours. In case of a failure in the long time running the user is offered compensation. Also, the management would like to have an estimate of probability of normal running without having to ask the process participants to avoid poor performance. In this context, we investigate two issues: exploring the failure paths, and estimating the probability of dead states in business processes.

Performance evaluation in service oriented process can be approached from a Petri Net modeling perspective. Timed Net Evaluation Tool (TimeNET) [55], is a software package for the modeling and evaluation of SPNs with non-exponentially distributed firing times. It provides a graphic interface and several specialized functions for the analysis and simulation of NMSTPN. For the structural analysis, the

<table>
<thead>
<tr>
<th>Marking</th>
<th>Places</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_0$</td>
<td>$P_6$  $P_7$ $P_8$ $P_9$ $P_{10}$ $P_{11}$ $P_{12}$ $P_{13}$ $P_{14}$ $P_{15}$ $P_{16}$ $P_{17}$ $P_{18}$</td>
</tr>
<tr>
<td>$M_1$</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_2$</td>
<td>0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_3$</td>
<td>0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_4$</td>
<td>0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_5$</td>
<td>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>$M_6$</td>
<td>0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_7$</td>
<td>0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_8$</td>
<td>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>$M_9$</td>
<td>0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>$M_{10}$</td>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>$M_{11}$</td>
<td>0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{12}$</td>
<td>0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>$M_{13}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>$M_{14}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>$M_{15}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{16}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>$M_{17}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>$M_{18}$</td>
<td>0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{19}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{20}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{21}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{22}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{23}$</td>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{24}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$M_{25}$</td>
<td>0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Table II. List of the markings for reachability tree shown in Fig. 8.

<table>
<thead>
<tr>
<th>No.</th>
<th>Failure Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$T_0T_2T_3T_{17}$</td>
</tr>
<tr>
<td>2</td>
<td>$T_0T_3T_{16}$</td>
</tr>
<tr>
<td>3</td>
<td>$T_0T_3T_5T_{10}T_{12}$</td>
</tr>
<tr>
<td>4</td>
<td>$T_0T_3T_5T_{10}T_{12}$, $T_0T_3T_5T_{10}T_{16}$, $T_0T_3T_5T_{16}$</td>
</tr>
<tr>
<td>5</td>
<td>$T_0T_3T_5T_{17}$, $T_0T_3T_5T_{17}T_{3}$</td>
</tr>
<tr>
<td>6</td>
<td>$T_0T_3T_5T_{10}T_3T_{12}$, $T_0T_3T_5T_{10}T_3T_{16}$, $T_0T_3T_5T_{10}T_{16}$</td>
</tr>
<tr>
<td>7</td>
<td>$T_0T_3T_5T_{10}T_3T_{12}$, $T_0T_3T_5T_{10}T_3T_{16}$, $T_0T_3T_5T_{10}T_{16}$</td>
</tr>
<tr>
<td>8</td>
<td>$T_0T_3T_5T_{17}$, $T_0T_3T_5T_{17}T_{3}$, $T_0T_3T_5T_{17}T_{3}$</td>
</tr>
<tr>
<td>9</td>
<td>$T_0T_3T_5T_{10}T_3T_{12}$, $T_0T_3T_5T_{10}T_3T_{12}$, $T_0T_3T_5T_{10}T_{16}$, $T_0T_3T_5T_{10}T_{16}$, $T_0T_3T_5T_{10}T_{16}$</td>
</tr>
<tr>
<td>10</td>
<td>$T_0T_3T_5T_3T_3T_{12}$, $T_0T_3T_5T_3T_3T_{12}$, $T_0T_3T_5T_3T_3T_{12}$</td>
</tr>
<tr>
<td>11</td>
<td>$T_0T_3T_5T_3T_3T_{12}$, $T_0T_3T_5T_3T_3T_{12}$, $T_0T_3T_5T_3T_3T_{12}$</td>
</tr>
<tr>
<td>12</td>
<td>$T_0T_3T_5T_3T_3T_{12}$, $T_0T_3T_5T_3T_3T_{12}$, $T_0T_3T_5T_3T_3T_{12}$</td>
</tr>
</tbody>
</table>

Table III. Sequences of transitions that lead to failures.
The sequences when the failure occurs and the second column presents the failure paths.

### 5.2 Estimation of Failure Rates

NHPP model has been used in practical systems based on input parameters. Given a failure data set of a web service, we use Statistical Product and Service Solutions to estimate the three input parameters: the total number of initial faults $a$, the fault detection rate $b$, and the shape factor $d$. They could also be obtained using least squares method, method of moments, maximum likelihood, and Bayesian methods [56], [57], [58]. We employ earlier white-noise model [47], [49] to estimate the correlation strength $\sigma$. It is noted here that some the web services are in a more open network and hence are prone to network uncertainty; whereas the others describe a closed environment. Such differences of environmental uncertainty would directly affect the fault detection process, and hence we prescribe larger values of $\sigma$ for the web services in open network.

We use least squares method to estimate the three parameters. The main idea is that the overall solution minimizes the sum of the squares of the errors made in the result of the equation. The basis of the method is to approximate the model by a linear one and to refine the parameters by successive iterations.

Following the above approach, the failure rates of $T_{12}$, $T_{13}$, $T_{14}$, $T_{15}$, $T_{16}$, and $T_{17}$ can be derived from NHPP model, based on the data of the development phase. By setting the parameters of NHPP model, the failure rates of the six services can be obtained, as depicted in table IV.

### 5.3 Qualitative Performance Assessment

To validate our framework, we implement the systematic performance evaluation procedure based on the failure paths. The failure sequences of events or markings for the transitions mean that the system would go into the failure state by firing through these sequences, which are listed in table III. As soon as the failure sequences of transitions have been obtained, it is necessary to calculate the transitional probabilities. We take the sequences number 1 as an example to demonstrate the procedure of determining the transitional probability. We want to calculate the probability that the TP service will fail in $200h$, with $\beta = 1$. The weights $w_i$ of all transitions in sequences number 1 is listed in table V.
Table V. Weights of transition in failure sequence # 1

<table>
<thead>
<tr>
<th>w_i</th>
<th>w_i</th>
<th>w_i</th>
<th>w_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3331</td>
<td>0.3331</td>
<td>0.3331</td>
<td>6.3293 × 10^{-4}</td>
</tr>
</tbody>
</table>

Table VI. Time of transition in failure sequence # 1

<table>
<thead>
<tr>
<th>( \tau_i ) (h)</th>
<th>( \tau_i ) (h)</th>
<th>( \tau_i ) (h)</th>
<th>( \tau_i ) (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>66.6245</td>
<td>66.6245</td>
<td>66.6245</td>
<td>0.1266</td>
</tr>
</tbody>
</table>

The approximate interval which includes a lower and an upper bound can be computed, as listed in Table VI. The lower bound is the time when the transition would be able to fire and the upper bound is the time when the same transition is fired.

So far, we have all information available to calculate the transitional probabilities for each failure sequence:

\[
F_i(x) = \int_0^x \lambda_i \beta x^{\beta-1} e^{-\lambda_i x^{\beta}} dx = 1
\]

\[
F_2(x) = \int_0^x \lambda_i \beta x^{\beta-1} e^{-\lambda_i x^{\beta}} dx = 1
\]

\[
F_3(x) = \int_0^x \lambda_i \beta x^{\beta-1} e^{-\lambda_i x^{\beta}} dx = 1
\]

\[
F_4(x) = \int_0^x \lambda_i \beta x^{\beta-1} e^{-\lambda_i x^{\beta}} dx = 2.4049 \times 10^{-4}
\]

As presented in Table VII, the superscript \( k \) in \( F_k \) is an index for identifying the location in the sequences resulting in the failure of the TP process, and \( F_k \) means the probability of marking starting in \( i \) and ending in \( j \), in a given sequence \( k \). The probability \( F_k \) can be computed which reveals the failure probability of each failure sequence. Based on the theory defined above, the probability that the TP process would not fail in 200h is computed as \( P = 1 - F = 1 - 0.0027 = 0.9973 \).

The results of the generated failure probability with the quantitative solution may be used in design and maintenance of a service-oriented application of the enterprises.

5.4 Sensitivity Analysis

By subsuming our previous results, we answer questions frequently raised by software engineers at the design/implementation phases of service-oriented applications, such as: “what is the system performance degradation rate?”. The answer for these questions are mostly unclear at design/implementation steps and deriving them using purely empirical evidences may lead to unreliable action, such as the distribution of resources and strategy. A contribution of the paper is a framework for estimating the performance in practice.

Due to the situation that most business processes take a certain time or sometimes even months to complete, we now conduct a sensitivity analysis to study the impact of different types of distributions (Weibull, Normal, Uniform, and Lognormal distributions) on the performance of the prediction approach with respect to running time.

Table VII. Performance of sequences of transitions that lead to failures

<table>
<thead>
<tr>
<th>No</th>
<th>Equations</th>
<th>Failure probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( F_i = \prod_i F_i(x)_i )</td>
<td>2.4049 × 10^{-4}</td>
</tr>
<tr>
<td>2</td>
<td>( F_i = \prod_i F_i(x)_i )</td>
<td>1.1261 × 10^{-4}</td>
</tr>
<tr>
<td>3</td>
<td>( F_i = \prod_i F_i(x)_i )</td>
<td>2.4180 × 10^{-4}</td>
</tr>
<tr>
<td>4</td>
<td>( F_i = 3 \times \prod_i F_i(x)_i )</td>
<td>2.5341 × 10^{-4}</td>
</tr>
<tr>
<td>5</td>
<td>( F_i = 3 \times \prod_i F_i(x)_i )</td>
<td>5.4119 × 10^{-4}</td>
</tr>
<tr>
<td>6</td>
<td>( F_i = 3 \times \prod_i F_i(x)_i )</td>
<td>2.0274 × 10^{-4}</td>
</tr>
<tr>
<td>7</td>
<td>( F_i = 3 \times \prod_i F_i(x)_i )</td>
<td>5.8049 × 10^{-4}</td>
</tr>
<tr>
<td>8</td>
<td>( F_i = 6 \times \prod_i F_i(x)_i )</td>
<td>9.1355 × 10^{-4}</td>
</tr>
<tr>
<td>9</td>
<td>( F_i = 6 \times \prod_i F_i(x)_i )</td>
<td>7.8407 × 10^{-4}</td>
</tr>
<tr>
<td>10</td>
<td>( F_i = 3 \times \prod_i F_i(x)_i )</td>
<td>1.4483 × 10^{-4}</td>
</tr>
<tr>
<td>11</td>
<td>( F_i = 3 \times \prod_i F_i(x)_i )</td>
<td>1.9196 × 10^{-4}</td>
</tr>
<tr>
<td>12</td>
<td>( F_i = 3 \times \prod_i F_i(x)_i )</td>
<td>1.6871 × 10^{-4}</td>
</tr>
</tbody>
</table>

Figure 9 presents the behavioral plots and computational results of four models with respect to TP service. Similar trends are exhibited for all four distributions: large probability at the earlier stage but gradual decreases at later time. This is expected, since few failure data are available at earlier time; as time elapses and more faults can be detected, its performance decreases. Overall, the Weibull approach yields the smallest rate of decline. Based on the assumption that the time from occurrence of failure is negligible in comparison to activity duration, next we conduct further numerical evaluations. For validation, we compare the elapsed time as conditioning factor on the duration distributions (Weibull, Normal, Uniform, and Lognormal distributions) with the baseline model (Markov chain).

![Figure 9. Sensitivity analysis of Weibull, Normal, Uniform, and Lognormal distributions on TP service with various configurations of running time](image-url)
5.4.1 Mean Square Error (MSE)

\[ \text{MSE} = \frac{1}{K} \sum_{i=1}^{K} \left[ (M(t_i)) - q(t_i) \right]^2 \]

Here, \( K \) denotes observation points and \( q(t_i) \) is the observed faults at \( t_i \). A lower value of MSE means a good prediction.

5.4.2 Root Mean Square Prediction Error (RMSPE)

\[ \text{RMSPE} = \sqrt{\text{Bias}^2 + \text{PRV}^2} \]

RMSPE contains two additional indicators: the average of prediction errors (Bias) and the predicted relative variation (PRV).

\[ \text{Bias} = \frac{1}{K} \sum_{i=1}^{K} \left[ (M(t_i)) - q(t_i) \right] \]

\[ \text{PRV} = \frac{\sqrt{\sum_{i=1}^{K} \left[ (M(t_i)) - q(t_i) - \text{Bias} \right]^2}}{K-1} \]

Lower absolute values of Bias, PRV and RMSPE represent a close match to the data.

Figure 10 shows the MSE, Bias, PRV and RMSPE results of different types of distributions, compared with the baseline model, in estimating the probability that reflects transgressing the failure state of the TP service. It confirms that the Weibull yields the best simulation of the exponential baseline model, and hence can approximate the result of Markov chain technique in reality. Comparing with the Markov chain technique, our NMSTPN model assuming non-Markovian distributions overall tends to overestimate the probability in this case and these deviations gradual increase as time elapse. This can be well explained by many cases in the predictions, as the failing duration may follow different distributions in real applications [59], [60]. Overall, the proposed approaches go beyond conventional exponential technique. By employing a non-Markovian (non-exponential) distribution, one is enabled to evaluate probabilities of application under different assumptions, which is crucial in system performance assessment. The purpose of the method is to analyze and improve the performance of the application and thus based on the behavioral plots, the system designer can visualize the failure sequences of the system and plan for suitable optimization and design.

6. CONCLUSION
The assessment of system performance with failures is essential in improving the overall quality of the service-oriented applications, which increases the usability of the applications and enhances the core-competitiveness of an enterprise. Current techniques for analyzing the performance of applications are lack of failure data and without the consideration that failure times may be (not limited to) a non-Markovian (non-exponential) distribution. Therefore, in this paper we present the new technique to estimate the failure rate and method for computing the probability of the sequence of failures, hence, overcome the limitations. The following set of strategies are presented: a NHPP model is applied to estimate the failure rate; NMSTPN model is defined to describe the application dynamics; based on reachability tree derived from NMSTPN model, the sequences of failures can be explored; and the probabilities of the sequences of failures are computed from transitions with non-exponentially distributed ring times. An example of a TP service is presented to demonstrate the concept. As an extension of the traditional methods, the resulting framework can be used as a systematic approach for designers to evaluate the performance of complex service-oriented applications. Data can be applied to help the enterprises to meet performance requirements.

We argue that the proposed approach can be extended to other fields of interest and open the door for innovative research to keep on improving the scalability and performance of current solutions. As for future work, we would like to evaluate our approach with more complex applications.

7. REFERENCES


Authors

Jiajun Xu is currently pursuing his Ph.D. in Computer Science and Application at Beihang University. He received his B.S. degree at George Mason University and Wuhan Tech in 2008. He obtained his M.S. degree from Beihang University in 2011. His research areas include: Petri Net Theory and Application, Service-oriented Computing, Software Reliability, and Uncertainty Quantification.

Shuzhen Yao is a Professor in the School of Computer Science and Engineering at Beihang University. She received her BS, Master and PhD degrees in Computer Science from Beihang University in 1986, 1989, and 2008, respectively. She was previously an academic visitor from 2005 to 2006 at the University of Illinois at Chicago. Her research interests include: Petri Net Theory, Software Development Environment, Software Reliability, and Network Security.