7-2015

Reliability assessment for distributed systems via communication abstraction and refinement

Lin GUI

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Yang LIU

Jin Song DONG

Citation

GUI, Lin; SUN, Jun; LIU, Yang; and DONG, Jin Song. Reliability assessment for distributed systems via communication abstraction and refinement. (2015). *Proceedings of the 2015 International Symposium on Software Testing and Analysis, Baltimore, USA, July 13-17*. 293-304.
Available at: https://ink.library.smu.edu.sg/sis_research/4955

Reliability Assessment for Distributed Systems via Communication Abstraction and Refinement

Lin Gui*, Jun Sun†, Yang Liu‡, and Jin Song Dong*

*National University of Singapore, Singapore
†Singapore University of Technology and Design, Singapore
‡Nanyang Technological University, Singapore

ABSTRACT

Distributed systems like cloud-based services are ever more popular. Assessing the reliability of distributed systems is highly non-trivial. Particularly, the order of executions among distributed components adds a dimension of non-determinism, which invalidates existing reliability assessment methods based on Markov chains. Probabilistic model checking based on models like Markov decision processes is designed to deal with scenarios involving both probabilistic behavior (e.g., reliabilities of system components) and non-determinism. However, its application is currently limited by state space explosion, which makes reliability assessment of distributed system particularly difficult. In this work, we improve the probabilistic model checking through a method of abstraction and reduction, which controls the communications among system components and actively reduces the size of each component. We prove the soundness and completeness of the proposed approach. Through an implementation in a software toolkit and evaluations with several systems, we show that our approach often reduces the size of the state space by several orders of magnitude, while still producing sound and accurate assessment.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Model checking, Reliability*

General Terms

Reliability, Verification

Keywords

MDPs, reliability assessment, probabilistic model checking

1. INTRODUCTION

Reliability and fault tolerance are central to many distributed systems, including cloud-based web services, industrial control systems, wireless sensor networks, etc. The term reliability refers to the probability of failure-free software operation within a specified period of time and environment [41, 10]. Being reliable is the key for these systems, as failures would damage the reputation of the system operators, and potentially lead to losses in capitals or even human.

Assessing the reliability of a distributed system can be highly non-trivial due to non-determinism, in contrast to that of a sequential system, which is often deterministic. In distributed systems, the order of executions among various system components is highly unpredictable and dependent on the operating environment. Therefore, the precise probability distribution of execution orders is hard to obtain if not impossible, and it is more suitable to model them non-deterministically. However, non-determinism invalidates existing reliability assessment approaches based on Markov chain models [10, 37, 30, 28, 21], which fundamentally assume that there is a unique probability distribution of event occurrences at any system state. It is thus necessary to develop a method for accessing the reliability of non-deterministic systems.

A potential candidate is probabilistic model checking [6, 13] based on Markov decision processes, which is designed to deal with both probabilistic behavior and non-determinism. However, its application is limited to small scale distributed systems as it works by exhaustively exploring the global state space, which is the product of the state spaces of all components and often huge. Therefore, we are motivated to develop a scalable approach to assess the reliability of distributed systems (e.g., web services, wireless sensor network) that often consist of many components (e.g., clients, sensors).

In this work, we assume that a distributed system consists of a set of components, each of which has its local state space and interfaces for communications. Our key idea is to shrink the global state space by controlling the communications among system components and actively reducing the sizes of their local state spaces. More specifically, we start with an abstract system by turning a subset of communication events into local (i.e., non-communication) events, which can be effectively removed afterwards by re-calculating the local probability distributions for the rest of the communication events. We then perform probabilistic model checking to calculate the reliabilities (here referred to as 'approximations'). If the resulting approximations are not precise enough, refinement can be performed by incrementally enlarging the set of communication events, which eventually yield an actual result based on the complete model. We

ISSTA'15, July 13–17, 2015, Baltimore, MD, USA

http://dx.doi.org/10.1145/2771783.2771794

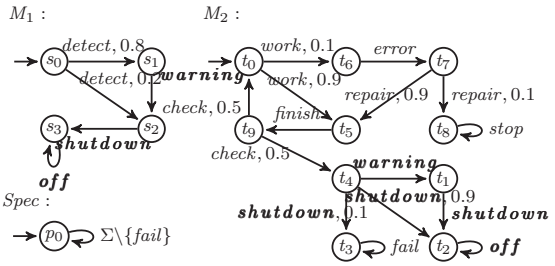

Figure 1: Markov models and an LTS specification

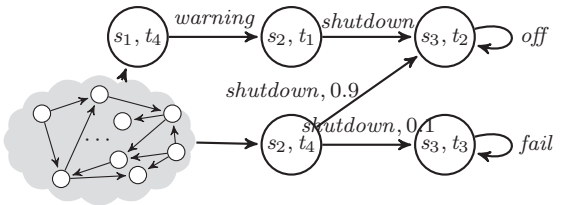

Figure 2: The state space of the product M_1 and M_2

prove the soundness of our approach by showing that probabilistic model checking on the reduced system always results in safe approximations and, more importantly, the approximations are often precise enough so that we can derive conclusive results based on a smaller state space. We implement the proposed abstraction and refinement framework in a toolkit RaPiD [31] to support the reliability assessment of distributed systems. Through several evaluations on multiple systems, results show that our method improves the performance significantly.

2. MOTIVATING EXAMPLE

A simple model of a device controlling system, which is a variant of [40], is shown in Fig. 1. The device is modeled as a Markov decision process (MDP) M_2. Its shutdown process is coordinated by a controller, modeled as a Markov chain (MC) M_1. Each probability distribution (PD) is labeled with an event name. For example of the PD at state t_0 in M_2, the transition from t_0 to t_6 (labeled with $work, 0.1$) means that while the device is working on a task, it has a probability of 0.1 going to state t_6. The controller and the device communicate through synchronizing common events as highlighted in bold. If no local transition is enabled, the controller and the device advances to the next state when this communication event occurs simultaneously in both the controller and the device. Intuitively, M_1 first receives a *detect* signal, after which it sends a *warning* message and coordinates the shutdown behavior of the device by sending a *shutdown* command. However, with a probability of 0.2, it fails to issue a *warning* message. If M_2 receives *warning*, it shuts down correctly; otherwise, it only shuts down correctly with a probability of 0.9. Events *warning* and *shutdown* are modeled non-deterministically at state t_4, as the exact probability of each event occurrence depends on control environment in practice.

A labeled transition system, *Spec* in Fig. 1, specifies a system that always shuts down properly without any occurrence of *fail* event. Here Σ is an abbreviation of the set of all events in M_1 and M_2, and the transition labeled with $\Sigma\backslash\{fail\}$ denotes a group of transitions that are labeled with

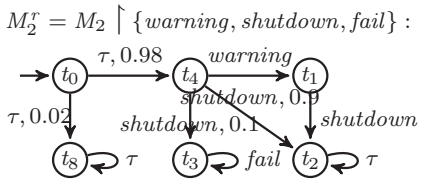

Figure 3: A reduced model by hiding local and *off* events

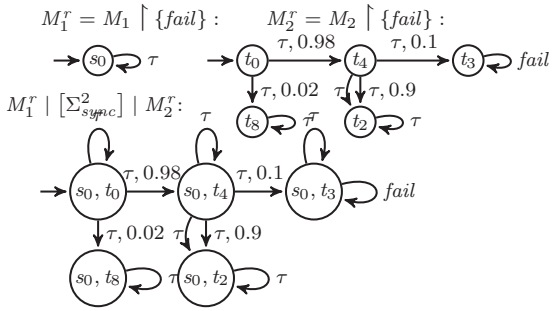

Figure 4: Reduced models with *fail* event visible

any event in Σ except *fail* event. The question is how reliable the system is for accomplishing a shutdown properly; or equivalently what the minimum probability is for the system to satisfy the *Spec*.

The standard approach works as follows. First, we compute the synchronous product of M_1 and M_2. Notice that all common events *warning*, *shutdown* and *off* are to be synchronized. They are referred to as communication events, and the rest are referred to as local events. The result is an MC partially shown in Fig. 2. There are in total 24 states and 47 transitions. Due to the space limit, we only show the part that contains the communication events. Next, we apply probabilistic model checking to the computation of the probability that the product satisfies *Spec*, which is reduced to the problem of computing the minimum probability of not reaching state (s_3, t_3) in the product [50]. Using standard techniques like value iteration or solving a linear equation system, we obtain a reliability of 0.9804.

An improved approach being proposed in this work is based on two major observations: (1) some of the communication events are not essential in computing this probability, and (2) in general, we can always identify lower and upper bounds of the reliability even if we choose to ignore some of the communications among the components. In the following, we illustrate our improved approach through two cases.

In the first case, we show that ignoring some of the communication events allows us to work with a smaller state space without changing the result. In fact, local events do not affect the communication among components, and communication event *off* is likely not that relevant to overall reliability intuitively, as both M_1 and M_2 remain at the same state after it occurs. Thus, we hide all local and *off* events by replacing them with an invisible event, designated by τ event. M_1 and M_2 are reduced to M_1^r, M_2^r, respectively. M_1^r is exactly the same as M_1 except that the self-loop transition labeled with *off* event previously at state s_3 is hidden and labeled with τ event instead. To further explain the reduction process, M_2^r is shown in Fig. 3. After hiding, states t_0, t_5, t_6, t_7, and t_9 are only connected by τ events and thus can

be collapsed into one state t_0 connecting to states t_4 and t_8 with an equivalent PD, which can be obtained via solving a simple linear program [48, 3]. The states t_0, t_1, t_2, t_3, t_4 and t_8 are kept in M_2^r as they cannot be further reduced. As a result, the number of states of the parallel composition is reduced from 24 to 13. With probabilistic model checking, the minimum probability is 0.9804 and the same as that of the original model.

In the second case, we show that a safe approximation of the minimum reliability can be obtained even though the 'wrong' events are ignored. We consider an extreme case by ignoring all local and communication events except *fail* event as *fail* is the only event related to *Spec*. The reduced models M_1^r and M_2^r are shown in Fig. 4. As all the events in M_1 are hidden, the whole model is reduced to one state with a self-looping τ transition M_1^r. Similar to the first case, the states t_5, t_6, t_7, and t_9 are removed in M_2^r. The transitions between the states t_4 and t_2, which are previously linked by events *shutdown* and *warn*, are now reduced into two direct τ transitions. The parallel composition of the two reduced models is shown in the third model of Fig. 4. Notice that the number of states of the parallel composition is reduced from 24 to 5. Furthermore, the product has no loops with multiple states so that probabilistic model checking with value iteration converges fast. Based on this highly abstracted model, the minimum probability is calculated as 0.9020, which is smaller than the actual value 0.9804, and thus a safe approximation. If the question is whether the system has a reliability of at least 0.9 shutting down successfully, we can conclude positively with this result. However, it should be noted that according to the actual requirement, refinement can always be performed to yield more accurate result.

This example provides insights on the effectiveness of our approach in speeding up the reliability assessment as well as potential challenges. In the following, we present details of our approach including the heuristics on choosing the 'right' events to ignore.

3. PRELIMINARIES

3.1 Markov Decision Processes

We start with defining a labeled transition system (LTS). Let Σ be a set of event names; τ denote an internal event that is invisible from external; and Σ_τ denote $\Sigma \cup \{\tau\}$. An LTS is a tuple $\mathcal{L} = (S, init, Act, T)$ where S is a set of states; $init \in S$ is the initial state; $Act \subseteq \Sigma_\tau$ is a set of events (or called an alphabet); and $T \subseteq S \times Act \times S$ is a labeled transition relation. A simple example is the *Spec* shown in Fig. 1.

Let $s, s' \in S$ and $a \in \Sigma_\tau$, a transition between two states s, s', is denoted as $(s, a, s') \in T$ or written as $s \xrightarrow{a} s'$ for simplicity. In this case, we say a is enabled at s. We write $u \rightsquigarrow v$ to denote that v is reachable from u through τ transitions, i.e., there exists a finite sequence of states $\langle s_0, s_1, \cdots, s_n \rangle$ such that $s_i \xrightarrow{\tau} s_{i+1}$ for all $i \in [0, n-1]$ and $u = s_0$ and $v = s_n$. We write $u \overset{a}{\rightsquigarrow} v$ if $u \rightsquigarrow u'$ and $u' \xrightarrow{a} v'$ and $v' \rightsquigarrow v$. This means that the two states are connected via a series of τ transitions and one a transition. A path of $\mathcal{L}$ is a sequence of alternating states/events $\pi = \langle s_0, a_0, s_1, a_1 \cdots \rangle$ such that $s_0 = init$ and $s_i \xrightarrow{a_i} s_{i+1}$ for all $i \geq 0$. The set of all paths of $\mathcal{L}$ is written as $paths(\mathcal{L})$. Given a path π, we can obtain the corresponding trace, written as $trace(\pi)$, by omitting states and τ events. The traces of $\mathcal{L}$ are denoted as $traces(\mathcal{L}) = \{ trace(\pi) \,|\, \pi \in paths(\mathcal{L}) \}$. An LTS is deterministic iff for all $s \in S$ and $e \in \Sigma_\tau$, if $s \xrightarrow{e} u$ and $s \xrightarrow{e} v$, then $u = v$. Otherwise, it is non-deterministic. A non-deterministic LTS can be translated into a trace-equivalent deterministic LTS using the standard power set construction [47, 50].

Informally, an LTS can be turned into an MDP by incorporating probability distributions (PDs), e.g., to model system failures probabilistically. For instance, in M_1 as shown in Fig. 1, it assumes that the event *detect* is not perfectly reliable. Instead of having a simple event *detect*, we have a PD that with a probability of 0.8 the detection is successful (and a *warning* is generated afterwards) and with a probability of 0.2 that it fails. Formally, given a set of states S, a PD is a function $\mu : S \to [0, 1]$ such that $\Sigma_{s \in S} \mu(s) = 1$. Let $Distr(S)$ be the set of all PDs over S. An MDP is a tuple $\mathcal{M} = (S, init, Act, Pr)$ where S is a set of states; $init \in S$ is the initial state; Act is an alphabet; and $Pr : S \times Act \to Distr(S)$ is a labeled transition relation. An example is M_2 shown in Fig. 1. In this work, we assume that all MDPs are deadlock-free. It is known that a deadlocking state in an MDP can be replaced by a state that has a self-loop τ without affecting analysis results [6].

An MDP is non-deterministic if any state has more than one PD. A discrete time Markov chain (DTMC) can be viewed as a special MDP with every state having exactly one PD, and thus is deterministic. An example of a DTMC is M_1 shown in Fig. 1. An MDP $\mathcal{M}$ can be viewed as a group of DTMCs, each of which is obtained with a different *scheduler*. The scheduler, denoted as δ, selects an event (and the corresponding PD) at each state so that the result is a DTMC, denoted as $\delta(\mathcal{M})$. Formally, a scheduler is a function deciding which event to choose based on the execution history. A memoryless scheduler is a function $\delta : S \to Act$ that always chooses the same event given the same state. It has been shown that memoryless schedulers are sufficient for our present task [6]. Thus we focus on memoryless schedulers only.

Given a DTMC $\delta(\mathcal{M})$, a path is a sequence $\langle s_1, a_1, s_2, a_2, \cdots \rangle$ such that $a_i = \delta(s_i)$ is the event chosen by the scheduler δ and $Pr(s_i, a_i)(s_{i+1}) > 0$. For each path, we can calculate its probability as $\Pi_i Pr(s_i, a_i)(s_{i+1})$. Given the path, we can obtain a trace $\langle a_1, a_2, \cdots \rangle$ by omitting the states and τ events. The probability of a given trace tr, written as $Pr(tr, \delta(\mathcal{M}))$, is defined as the accumulated probability of all the paths in $\delta(\mathcal{M})$ that exhibit tr.

A distributed system with failure behavior can often be modeled as a network of MDPs. Given a system of multiple MDPs, events to be synchronized are called communication events that are the common events among the MDPs. Within a set of events, visible events are the ones that can be observed from outside, and the rest are called local or internal events. A communication event can be synchronized if and only if it is a visible event and is enabled in all MDPs. We define the synchronization of two MDPs over a set of visible events as follows, which can be readily extended to multiple MDPs.

Let $\mathcal{M}_i = (S_i, init_i, Act_i, Pr_i)$ where $i \in \{1, 2\}$ be two MDPs and Σ_v be a set of visible events. The synchronization composition $\mathcal{M}_1$ and $\mathcal{M}_2$ over Σ_v, written as $\mathcal{M}_1 \,|[\Sigma_v]|\, \mathcal{M}_2$, is an MDP $\mathcal{M} = (S_1 \times S_2, (init_1, init_2), Act_1 \cup Act_2, Pr)$, where Pr is the probability transition satisfying:

- if $s_1 \xrightarrow{e} \mu$ in Pr_1 and $e \notin \Sigma_v$, then $(s_1, s_2) \xrightarrow{e} \mu'$ in Pr for all $s_2 \in S_2$ such that $\mu'((s_1', s_2)) = \mu(s_1')$ for all $s_1' \in S_1$;
- if $s_2 \xrightarrow{e} \mu$ in Pr_2 and $e \notin \Sigma_v$, then $(s_1, s_2) \xrightarrow{e} \mu'$ in Pr for all $s_1 \in S_1$ such that $\mu'((s_1, s_2')) = \mu(s_2')$ for all $s_2' \in S_2$;
- if $s_1 \xrightarrow{e} \mu_1$ in Pr_1, $s_2 \xrightarrow{e} \mu_2$ in Pr_2 and $e \in \Sigma_v$, then $(s_1, s_2) \xrightarrow{e} \mu'$ in Pr such that $\mu'((s_1', s_2')) = \mu_1(s_1') \cdot \mu_2(s_2')$ for all $s_1' \in S_1$ and $s_2' \in S_2$.

Examples of the composition is shown in Fig. 2 and Fig. 4. We remark that, though only synchronous communication is allowed in the above definition, asynchronous communication, which is typical for distributed systems, can be easily constructed by modeling the communication media explicitly. For instance, if $\mathcal{M}_1$ and $\mathcal{M}_2$ communicate through radio, which is common for sensors, we can model the lossy channel using an MDP that essentially receives messages and later on either forgets about the messages or forwards them. The entire system is then a composition of the three MDPs.

3.2 Probabilistic Model Checking

A specification of correct system behavior can be modeled in either linear temporal logic [46] or labeling transition system (LTS). The reliability is then calculated as the probability of a system model (which is a network of MDPs) satisfying the specification. In the simplest case, the specification is the one that prevents the *fail* event from happening and allows other events repeatedly to happen at any time, i.e., the LTS *Spec* shown in Fig. 1. Having a more general specification (which models the behavior of a perfectly reliable system according to different service requirements) allows more flexibility than always having the same one, *Spec*, as the specification (which specifies that the system should not *fail*). For instance, the specification can model a system that fails once but successfully activates a backup service, or a system that works correctly for important system functionalities whereas fails and recovers only during less important missions, etc. Furthermore, different systems may have different focuses with respect to the reliability. For instance, reliability of a sensor in a wireless sensor network may be defined as the probability of detecting certain phenomena, whereas the reliability of a web server is associated with the probability that it reacts to web page requests without errors.

The task of reliability assessment is thus to calculate the exact probability of a system satisfying the specification. That is, reliability is the probability of the traces of the system model being a subset of those of the specification. Formally, let $\mathcal{M}$ be the system model with failure behavior and $\mathcal{S}$ be the specification. Reliability is the accumulated probability of all traces of $\mathcal{M}$ that are also traces of $\mathcal{S}$. Let δ be a scheduler of $\mathcal{M}$, reliability with δ written as $\mathcal{R}(\delta(\mathcal{M}), \mathcal{S})$ is defined as: $\mathcal{R}(\delta(\mathcal{M}), \mathcal{S}) = \Sigma\{Pr(tr, \delta(\mathcal{M})) \mid tr \models traces(\mathcal{S})\}$. Notice that with different schedulers, the probability is often different and there are potentially infinitely many schedulers. Therefore the measurement of interest are the minimum and maximum probabilities, which are defined as: $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) = \inf_\delta \mathcal{R}(\delta(\mathcal{M}), \mathcal{S})$; $\mathcal{R}^{max}(\mathcal{M}, \mathcal{S}) = \sup_\delta \mathcal{R}(\delta(\mathcal{M}), \mathcal{S})$. Using the approach proposed in [50], the range of reliability can be calculated using the following steps. Taking LTS $\mathcal{S}$ as a specification, we first construct a deterministic LTS, denoted as $d(\mathcal{S})$, which is equivalent to $\mathcal{S}$ using the standard power set construction [47]. Next, we compute the synchronous product of $\mathcal{M}$ and $d(\mathcal{S})$, which is defined as follows. Let $\mathcal{M} = (S_\mathcal{M}, init_\mathcal{M}, Act_\mathcal{M}, Pr_\mathcal{M})$ and $d(\mathcal{S}) = (S_\mathcal{S}, init_\mathcal{S}, Act_\mathcal{S}, T_\mathcal{S})$. The product, written as $\mathcal{M} \times d(\mathcal{S})$, is an MDP $(S_\mathcal{M} \times S_\mathcal{S}, (init_\mathcal{M}, init_\mathcal{S}), Act_\mathcal{M} \cup Act_\mathcal{S}, Pr)$, where Pr is defined as follows:

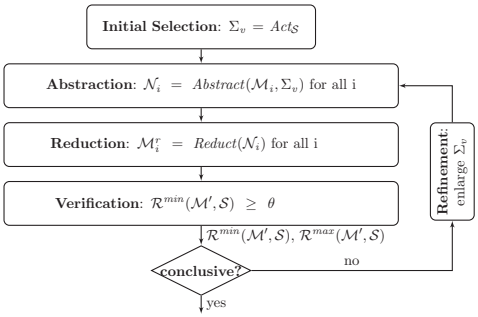

Figure 5: Overall workflow

- if $s_m \xrightarrow{e} \mu$ in $Pr_\mathcal{M}$ and $e \notin Act_\mathcal{S}$, then $(s_m, s_s) \xrightarrow{e} \mu'$ in Pr for all $s_s \in S_\mathcal{S}$ such that $\mu'((s_m', s_s)) = \mu(s_m')$ for all $s_m' \in S_\mathcal{M}$;
- if $s_m \xrightarrow{e} \mu$ in $Pr_\mathcal{M}$, $e \in Act_\mathcal{S}$ and $s_s \xrightarrow{e} s_s'$ in $d(\mathcal{S})$, then $(s_m, s_s) \xrightarrow{e} \mu'$ in Pr for all $s_s \in S_\mathcal{S}$ such that $\mu'((s_m', s_s')) = \mu(s_m')$ for all $s_m' \in S_\mathcal{M}$.

Here, $Act_\mathcal{S}$ is the set of specification events. Given a state $(s_m, s_s) \in S_\mathcal{M} \times S_\mathcal{S}$, if there exists $e \in Act_\mathcal{S}$ such that $s_m \xrightarrow{e} \mu$ in $\mathcal{M}$ and e is not enabled at s_s, any trace of $\mathcal{M}$ reaching s_m extended with e will not be a trace of $\mathcal{S}$, i.e., a trace example of $\mathcal{M}$ not satisfying $\mathcal{S}$. Thus, we call such states *witness states*. The reliability assessment problem is thus reduced to find the probability of reaching any witness state. For instance, if the maximum probability of reaching the witness states is q, then $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S})$ is $1 - q$. There are known methods for probabilistic reachability analysis like value iteration or solving linear programs [6]. Value iteration is an iterative approximation method and often has better scalability than other methods [39, 49]. However this approach converges much slowly for an MDP of large loops. Furthermore when the iteration is stopped, it is hard to know how far the approximation is from the actual result [9, 22].

4. OUR APPROACH

There are two major challenges in applying probabilistic model checking to reliability assessment of distributed systems. One is state space explosion, as the global state space is the product of that of all distributed components. The other is that the result is often an approximation without knowing the discrepancy from the actual result due to the limitation of value iteration. In this section, we present our approach, which aims to tackle the former and help the latter challenges by reducing states and loops. We assume that MDP is deadlock-free[1]. We present our approach by adopting a deadlock-free MDP $\mathcal{M} = (S_\mathcal{M}, init_\mathcal{M}, Act_\mathcal{M}, Pr_\mathcal{M})$ as the system model and an LTS $\mathcal{S} = (S_\mathcal{S}, init_\mathcal{S}, Act_\mathcal{S}, T_\mathcal{S})$ as the specification.

[1]This is a standard assumption. Nonetheless, if $\mathcal{M}$ is the parallel composition of multiple components, even if the components are deadlock-free, there is no guarantee that the composition is. On the other hand, there are methods that allow us to construct deadlock-free systems from given components [29].

4.1 Overview

The problem of reliability assessment for distributed systems is often stated as follows. Given a distributed system with multiple components, it is to decide whether the overall reliability is no less than some threshold θ. Thus, without loss of generality, we assume that we need to check whether $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ in the following. The workflow of our approach is shown in Fig. 5. Recall that Σ_v denotes the set of visible events and $Act_{\mathcal{S}}$ denotes the set of specification events. Any event not in Σ_v can be turned into a τ event. During **initial selection**, we always initialize Σ_v to $Act_{\mathcal{S}}$. Because hiding any event in the specification would invalidate the verification results. The other four main steps are detailed as follows.

In the **abstraction** step, we hide any event that is in $Act_{\mathcal{M}}$ but not in Σ_v. Specifically, for each component $\mathcal{M}_i$ in $\mathcal{M}$, if an event a is to be hidden, we label all transitions with τ instead of a in $\mathcal{M}_i$. As a result, if two MDPs communicate by event a, then the communication is lost. This alters the behavior of the system. Nonetheless, as we show later, this allows us to work with a reduced global state space after the reduction while being able to produce useful results.

In the **reduction** step, we minimize each component $\mathcal{M}_i$ by removing transitions labeled with τ in a way such that the verification result is not affected. We remark that, as the global state space is the product of the local state spaces, minimizing the local spaces would often lead to a significant reduction in the global state space.

In the **verification** step, we apply probabilistic model checking to check whether $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$ is true, where $\mathcal{M}'$ is the synchronization composition based on all components after abstraction and reduction. In fact, $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S})$ is always less than or equal to $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S})$ and $\mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$ is always larger than or equal to $\mathcal{R}^{max}(\mathcal{M}, \mathcal{S})$, as we will prove in Theorem 1.

After the verification step, we decide whether the result is conclusive. If $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$ is shown to be true, we conclude that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ is true and we are done with the assessment. If $\mathcal{R}^{max}(\mathcal{M}', \mathcal{S}) \leq \theta$, then we conclude that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ is false. Otherwise, the result is inconclusive, and we need to refine the abstraction so that more precise and perhaps conclusive results will be obtained.

In the **refinement** step, we reduce the set of events to be hidden by restoring some communication events. That is, we enlarge Σ_v so that the resultant system model $\mathcal{M}'$ is refined towards the original model $\mathcal{M}$. Afterwards, we repeat from the abstraction step. If all communication events are in Σ_v, the verification result of the last iteration is the final conclusion.

Our approach always terminates. The worst case is that, after a few rounds of abstraction/refinement, it terminates when all communication events are made visible. In our experiments, we show that our approach often produces relatively accurate results and concludes in early rounds. Even in the worst case, there can still be a reduction as we always hide the non-communication events. In the following, we present details of the key steps.

4.2 Abstraction and Reduction

In this part, we show how to systematically hide events in a system component and then build a reduced MDP that preserves the verification result on the overall system.

Given a component modeled as an MDP, denoted as $\mathcal{M}_i$ and a set of visible events Σ_v, the abstraction is done straightforwardly by turning transition labels not in Σ_v into τ. Let $\mathcal{N}_i$ denote the MDP of the component *after abstraction*. In the reduction step, we build a new MDP $\mathcal{M}_i^r$ such that probabilistic analysis results based on $\mathcal{N}_i$ are preserved in $\mathcal{M}_i^r$ (which could be different from that based on the original system component model obviously). The basic idea of the reduction is to remove τ transitions and group states which are connected by τ transitions (since they are indistinguishable from an external point of view).

Let $\mathcal{N}_i = (S, init, \Sigma_v \cup \{\tau\}, Pr)$. We formally define $\mathcal{M}_i^r$ as an MDP $(S', init, \Sigma_v, Pr')$ satisfying the following two conditions. First, $S' = \{init\} \cup \{s_i \in S \mid \exists\, s_j \in S, e \in \Sigma_v \cdot Pr(s_i, e)(s_j) > 0\}$ such that it contains the initial state of $\mathcal{N}_i$ and all states in S which have at least one outgoing transition labeled with a visible event. Intuitively, all other states in S are not in S' as all of their outgoing transitions are labeled with τ, and thus can be collapsed into some state in S'. Second, Pr' satisfies the following condition: if $s_i \in S'$, for all states $s_j \in S'$ such that there exists a scheduler δ that $s_i \overset{e}{\rightsquigarrow} s_j$ in $\delta(\mathcal{N}_i)$ where $e \in \Sigma_v$, then $Pr'(s_i, e)(s_j) = Pr(reach(s_i, s_j), \delta(\mathcal{N}_i))$. Here $Pr(reach(s_i, s_j), \delta(\mathcal{N}_i))$ is the probability of reaching state s_j from state s_i in a DTMC $\delta(\mathcal{N}_i)$. Calculating the probability of reaching a certain state in DTMC is a known problem with efficient solutions [6].

In the following, we discuss the complexity of the reduction defined above, especially in constructing Pr'. If $\mathcal{N}_i$ is a DTMC rather than an MDP, there is only a single scheduler to consider and the above construction is relatively inexpensive, as experimentally evidenced in [4, 3, 48, 32]. If $\mathcal{N}_i$ is an MDP[2], in constructing Pr', e.g., in identifying $Pr'(s_i, e)(s_j)$, we must explore all memoryless schedulers which result in DTMCs containing s_i and s_j. In the worst case, the number of such schedulers is exponential to the number of non-deterministic choices. This implies that the number of schedulers may remain the same though the state space is reduced.

The above defines the maximum reduction that we could achieve by eliminating all states whose outgoing transitions are labeled with τ. It is in general expensive to obtain the maximum reduction due to the large number of schedulers given a complicated MDP [17]. In order to obtain a reasonable reduction without paying the full price, our implementation for an MDP reduction focuses on two aspects. First, we identify all strongly connected components (SCCs) in $\mathcal{N}_i$ which only contain τ transitions and collapse all the states in each SCC into one representative. Recall that slow convergence in the value iteration method is often due to loops and a loop containing only τ transitions in $\mathcal{N}_i$ will result in many loops in the global space. Second, we remove states with all non-probabilistic outgoing transitions labeled with τ which are not part of any loop, and direct all their incoming transitions to the respective successors.

We remark that, a component-based reduction not only eases the state space explosion (since the number of states in each component may be reduced) but is also helpful in solving the slow convergence problem in probabilistic model checking. The reduction cost is relatively small compared to the potential saving due to the facts that (1) the size of a

[2] $\mathcal{N}_i^r$ is a probabilistic automaton rather than an MDP. In this work, the difference is irrelevant for simplicity in presentation.

local state space corresponding to a distributed component is relatively small, thus requires a relatively low reduction cost; and (2) any reduction on a local state space can produce exponential benefits when performing calculations in the global state space as many states and loops may have been eliminated. This is evidenced by the empirical evaluation in Section 5.

4.3 Verification

After abstraction and reduction, we apply probabilistic model checking to the analysis on whether $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$ is true or not, where $\mathcal{M}'$ is the synchronization composition of all the system components after abstraction and reduction. In the following, we discuss the effectiveness and the soundness of the abstraction and reduction by establishing that the verification result obtained based on $\mathcal{M}'$ is safe.

Let A be the alphabet of specification $\mathcal{S}$. Let G and L represent sets of communication and local events in $\mathcal{M}$. For each component, there are $2^{|A \cup G \cup L|}$ states in the worst case. Because for each event, there could be two states: either it is enabled or not. Assume there are X components running concurrently. In the worst case, there are $2^{|A \cup G \cup L| \times X}$ states. Given the set of visible events is Σ_v, the worst case number of states in the abstract model is $2^{|\Sigma_v| \times X}$, thus $2^{|(A \cup G \cup L - \Sigma_v|) \times X}$ states are reduced. In the extreme case, $\Sigma_v = A$ (i.e., all events not in A are hidden) and the worst case state space is $2^{|A| \times X}$, which is exponentially smaller than the original state space. We remark that the above state space calculation is only an estimate based on the assumption that states can be distinguished by their outgoing transitions. In the setting of an MDP, because the same event may be associated with different PDs, the resultant worst case state space could be even larger and so could the reduction. Nevertheless, the above analysis can provide some insights on how the proposed method may significantly improve reliability assessment for distributed systems.

THEOREM 1. *Let $\mathcal{M}$ be a deadlock-free MDP and $\mathcal{M}'$ be the abstract reduced MDP as described in Section 4.2.* $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \leq \mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$.

Proof It is trivial to see that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{M}, \mathcal{S})$ and thus in the following, we show $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \leq \mathcal{R}^{min}(\mathcal{M}, \mathcal{S})$ and $\mathcal{R}^{max}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$. Furthermore, extending the proof that a DTMC after reduction is equivalent to the original DTMC in probability measurement [48, 3], we can show that parallel composition of all components $\mathcal{N}_i$ (i.e., the component after abstraction) and that of all components $\mathcal{M}_i^r$ are equivalent. That is, assuming $\mathcal{N}$ is the parallel composition of $\mathcal{N}_i$, $\mathcal{R}^{min}(\mathcal{N}, \mathcal{S}) = \mathcal{R}^{min}(\mathcal{M}', \mathcal{S})$ and $\mathcal{R}^{max}(\mathcal{N}, \mathcal{S}) = \mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$. Thus, we are left with proving that the abstraction step is safe, i.e., $\mathcal{R}^{min}(\mathcal{N}, \mathcal{S}) \leq \mathcal{R}^{min}(\mathcal{M}, \mathcal{S})$ and $\mathcal{R}^{max}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{N}, \mathcal{S})$.

For simplicity, $\mathcal{M}$ is assumed to be the parallel composition of two components $\mathcal{M}_1$ and $\mathcal{M}_2$. It should be straightforward to extend the proof to multiple components. Let δ be an arbitrary memoryless scheduler for $\mathcal{M}$ and $\pi = \langle (s_0, t_0), a_0, (s_1, t_1), a_1, \cdots, a_{n-1}, (s_n, t_n) \rangle$ be a path of $\delta(\mathcal{M})$ where (s_i, t_i) is a state of $\mathcal{M}$. By an induction on the length of π, we show that there is always a scheduler δ' for $\mathcal{N}$ and a path $\pi' = \langle (s'_0, t'_0), a'_0, (s'_1, t'_1), a'_1, \cdots, a'_{n-1}, (s'_n, t'_n) \rangle$ of $\delta'(\mathcal{N})$ such that the probability of the two paths are the same and π and π' share the same sequence of events in $Act_{\mathcal{S}}$ and $s_n = s'_n$ and $t_n = t'_n$. The base case is when π is $\langle (s_0, t_0) \rangle$, which is trivially true. The induction hypothesis is that the above is true when π has n states. By assumption (that $\mathcal{M}$ is deadlock-free), there must be an event a_n such that $((s_n, t_n), a_n, \mu_{n+1})$ is a transition in $Pr_{\mathcal{M}}$. Here, let a_n be the choice of δ at (s_n, t_n). We discuss different cases.

- If $a_n \in \Sigma_v$, by definition, a_n must be enabled at (s'_n, t'_n) and it leads to the same PD. Therefore, we set $\delta'((s'_n, t'_n))$ to be a_n and the induction holds.
- If $a_n \notin \Sigma_v$ and a_n is a local event, there must be a τ transition enabled at (s'_n, t'_n) and it leads to the same probability distribution. Therefore, we set $\delta'((s'_n, t'_n))$ to be a_n and the induction holds.
- If $a_n \notin \Sigma_v$ and a_n is a communication event, there must be a transition labeled with a_n enabled at s_n leading to μ^1_{n+1} and a transition labeled with a_n enabled at t_n leading to μ^2_{n+1}. By definition of parallel composition, let the resultant product PD to be μ_{n+1}. There must be a τ transition enabled at s'_n leading to the same PD μ^1_{n+1} and there must be a τ transition enabled at t'_n leading to the same distribution μ^2_{n+1}. We set δ' such that $\delta'(s'_n, t'_n)$ is the τ transition at s'_n leading to μ^1_{n+1}. Next, for all states (s'_{n+1}, t'_n) such that $\mu^1_{n+1}(s'_{n+1}) > 0$, we set $\delta'(s'_{n+1}, t'_n)$ to be the τ transition at t'_n leading to μ^2_{n+1}. It can be shown that the following two paths: where $\frown$ denotes sequence concatenation,
  $\pi \frown \langle a_n, (s_{n+1}, t_{n+1}) \rangle$; $\pi' \frown \langle \tau, (s'_{n+1}, t'_n), \tau, (s'_{n+1}, t'_{n+1}) \rangle$
  have the same probability in $\delta(\mathcal{M})$ and $\delta'(\mathcal{N})$; and have the same sequence of events in $Act_{\mathcal{S}}$ (since a_n is not in Σ_v and therefore $Act_{\mathcal{S}}$); and have $s_{n+1} = s'_{n+1}$ and $t_{n+1} = t'_{n+1}$.

Thus, the scheduler space is enlarged after abstraction such that every scheduler in $\mathcal{M}$ will be still in $\mathcal{M}'$. Therefore, we conclude the induction holds. □

Based on the theorem, if $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$ is shown to be true, we conclude that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ is true and we are done with the assessment. If $\mathcal{R}^{max}(\mathcal{M}', \mathcal{S}) \leq \theta$, we conclude that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ is false. On the other hand, the usefulness of verification results based on $\mathcal{M}'$ depends on how close the results are to the verification results based on $\mathcal{M}$. In Section 5, we empirically show that they are indeed close and thus useful.

4.4 Refinement

In the refinement step, the set Σ_v is enlarged by adding communication events. Ideally, we should add events such that the size of the resultant system $\mathcal{M}'$ is small whereas the verification results are accurate. However, the computation of such a minimal alphabet is shown to be NP-hard. Instead, we propose two heuristics to automatically guide the refinement, which we demonstrate empirically to be effective.

With the first heuristic (hereafter $\mathcal{H}_1$), we always give higher priority to an event that is to be synchronously engaged by more components. Intuitively, if an event is to be synchronized by multiple components, adding it into Σ_v would not introduce many new states since the event is likely to be disabled most of the time. The purpose is to keep $\mathcal{M}'$ small.

With the second heuristic (hereafter $\mathcal{H}_2$), we evaluate and select the best group of events at each refinement iteration. Initially, we divide all candidate events into groups and each

group contains the same events in symmetrical components. Markov chain Monte Carlo sampling can be adopted to help group the events. This sampling method has been shown to be effective in practice for extremely large search spaces and is empirically known to converge well before the limit is reached [5]. In each refinement, we evaluate the effectiveness of the candidate groups and the events in the most effective group are then added to Σ_v. We quantify the effectiveness of every group, denoted by p, as follows,

$$p = w \cdot \underbrace{\frac{|\,\mathcal{R}(\mathcal{M}', \mathcal{S}) - \mathcal{R}(\mathcal{M}'_c, \mathcal{S})\,|}{\mathcal{R}(\mathcal{M}', \mathcal{S})}}_{p_a} + \underbrace{\mathrm{sgn}(N - N_c)\,\lg(\frac{|\,N - N_c\,|}{N})}_{p_s}$$

Here, $\mathcal{M}'$ and $\mathcal{M}'_c$ are the abstract reduced model before and after including the event into Σ_v; N and N_c are the number of states in $\mathcal{M}'$ and $\mathcal{M}'_c$; sgn() is the sign function. The part p_a measures improvement in the accuracy of a newly added event; and the other part p_s measures the changes in the size of state space. w is a weighting factor designed by users to control the degree of preference over the two aspects. We can observe that p measures the effectiveness of the newly added group reasonably well, as p increases when accuracy is improved and the number of states is reduced; and vice versa.

Compared with $\mathcal{H}_1$, $\mathcal{H}_2$ tends to provide a better performance because it is based on a more precise calculation. The price to pay is that $\mathcal{H}_2$ requires more effort on calculating the effectiveness factor p for each event group in every refinement step. $\mathcal{H}_1$ requires less as it only counts the number of system components that synchronize each event in the initial step.

Lastly, we argue that there are at most $L + 1$ refinement steps where L is the total number of communication events, and therefore our approach always terminates.

5. EXPERIMENTS AND EVALUATIONS

The proposed method has been implemented in RaPiD (Reli*a*bility Pred*i*ction and Distribution) [31] to support the reliability assessment of distributed systems, with 6.5K lines of C# code. RaPiD is a self-contained toolkit for reliability assessment and publicly available at [2]. It provides a user-friendly interface to draw MDP models as well as fully automated methods for reliability analysis. All models and evaluation results are available online at [1].

5.1 Case Studies

We evaluate RaPiD with three systems, two benchmarks from the literature and one real-world healthcare system developed jointly by our group and a hospital in Singapore [43].

The first system is a client-server system (CSS) [23, 19], which is a typical distributed protocol that partitions workloads between a service provider (called a server) and service requesters (called clients). The server and clients often communicate over a network. As the number of clients increases, the interactions between the clients and the server become more complicated. Performing reliability analysis of such a system requires extensive expertise and time. We build a CSS system model with one server and k clients, where k is an integer of at least 2. The resource is to be shared by the clients in a mutually exclusive way. Each client initially has a probability of 0.1 getting failure. In addition, we consider a variant that is slightly more complicated, denoted as CSSr. Besides the behavior in CSS, each client can be successfully repaired with a probability of 0.9. In both cases, the overall reliability to be estimated is the probability that any client can successfully access the resource and meanwhile no multiple clients are accessing the resource simultaneously. The specification used in this case study has multiple events like *cancel* and *granted* (instead of a single *failure*) which models a system where mutual exclusion is guaranteed perfectly.

The second system is an automatic gas station system (GSS) [34], consisting of one operator, n pumps, one queue for each pump and m customers. The operator handles payments and schedules the use of pumps. Each customer first goes to the operator and prepays a certain amount. Then, the customer will be randomly allocated to a pump. There is a queue of waiting customers at each pump. A pump must be activated before serving customers. Once filling finishes, the pump signals the operator with the amount of gas provided to the customer. Next, the operator calculates the balance and then gives the change back to the customer. We consider two kinds of failure behavior of each pump, i.e., in the beginning, there is a probability of 0.01 that a pump cannot be started successfully, and during the service, there is also a probability of 0.01 that the pump fails to deliver gas. Whenever a failure occurs, there is a corresponding maintenance that has a probability of 0.99 to rectify the pump system. We calculate the probability of providing at least 100 pumping services to the customers without any failure. The size of the system depends on the number of pumps in the station (n), and the number of customers simultaneously asking for the service (m). Noted that we calculate the reliability with consideration of different number of customers simultaneously asking for the service, and pumps available in the gas station.

The third system is a smart healthcare system (SHS). It provides assistance to mild dementia patients. The system has been deployed in a nursing home over half a year for a trial [43, 42]. The system has multiple sensors, an inference engine and reminders to monitor the patients' behavior and send out reminding message. The system reliability highly depends on reliabilities of its sensors and networks. In this case study, we model failure behavior of each sensor and network according to the data collected from the project engineers. In particular, RFID sensors has a probability of 0.75 not detecting the patients' ID (relatively high due to high sensitivity to distance); pressure sensors have a probability of 0.02 not detecting the patients lying on the bed, and the Bluetooth having a probability of 0.30 not transmitting messages successfully. The PDs of event occurrences are closely related to the patients' behavior that are hard to predict, and thus modeled non-deterministically in this work. The reliability to be measured is the minimum probability that the alert is sent without any error when a patient is lying on the wrong bed.

5.2 Evaluation Results

The underlying assumptions of our approach are as follows: (1) by reducing each component, we could significantly reduce the state space hence assessment time, and (2) ignoring some of the communication events has limited impact on assessment results. In the following, we evaluate the assumptions and the reduction efficiency by answering three research questions.

RQ 1: How effective is our technique in terms of reducing the size of the state space and assessment time?

Table 1: Results of comparison between RaPiD and RaPiDr under different levels of abstraction

Case(*para.*)	(*para.*)	RaPiD w/o reductions		RaPiD with reductions (RaPiDr)							
				minimum abstraction				maximum abstraction			
		#States	Time (s)	#States $Ratio^\star$	Time $Ratio^\star$	$\mathcal{R}_{min}$	$\mathcal{R}_{max}$	#States $Ratio^\star$	Time $Ratio^\star$	$\mathcal{R}'_{min}$	$\mathcal{R}'_{max}$
GSS(n, m)	(1, 1)	1,428	0.21	1.17	0.98	0.98985	1.00000	3.52	0.96	0.98985	1.00000
	(1, 4)	99,760	38.18	1.42	9.18	0.98985	1.00000	98.87	351.14	0.98985	1.00000
	(1, 6)	728,100	290.68	1.39	7.80	0.98985	1.00000	516.02	1892.11	0.98985	1.00000
	(2, 1)	14,218	2.28	1.40	3.62	0.98975	1.00000	1.93	2.76	0.98965	1.00000
	(2, 4)	3,060,585	1202.35	1.41	4.39	0.98965	1.00000	81.76	546.69	0.98965	1.00000
	(2, 6)	OM	OM	OM	OM	OM	OM	∞	∞	0.98965	1.00000
	(3, 1)	100,000	23.89	1.85	9.44	0.98965	1.00000	1.83	9.96	0.98945	1.00000
	(3, 4)	OM	OM	OM	OM	OM	OM	∞	∞	0.98945	1.00000
CSS(k)	(2)	32	0.02	1.00	0.33	0.81000	0.99000	1.45	0.34	0.81000	0.99000
	(4)	682	0.02	1.00	0.21	0.65610	0.99990	2.07	0.14	0.65610	0.99990
	(8)	158,866	9.26	1.00	0.90	0.43047	1.00000	3.39	2.86	0.43047	1.00000
	(10)	2,052,094	187.12	1.00	1.13	0.34868	1.00000	4.07	5.29	0.34868	1.00000
CSSr(k)	(2)	45	0.02	1.41	0.04	0.97814	0.99988	2.05	0.04	0.97814	0.99988
	(4)	1,515	0.06	2.22	0.53	0.95676	1.00000	4.59	0.30	0.95676	1.00000
	(8)	1,031,735	77.58	6.49	7.69	0.91540	1.00000	22.03	24.68	0.91540	1.00000
	(10)	OM	OM	∞	∞	0.89539	1.00000	∞	∞	0.89539	1.00000
SHS		1,296,000	207.33	3.47	4.98	0.63274	1.00000	13.51	27.03	0	1.00000

$Ratio^\star = RaPiD/RaPiDr$

To answer this question, we compared the number of states in a system and the total time cost for assessment (including time spent on abstraction, reduction and verification) with/without use of reduction technique in RaPiD (referred to as RaPiD and RaPiDr, respectively). For cross referencing, we also compared RaPiDr with PRISM v4.0.3 [39] on an Intel(R) Xeon(R) CPU at 2.67 GHz with 12 GB RAM. The comparison results must be taken with a grain of salt as PRISM and RaPiD are designed for different purposes. We created several models with different sizes by varying the number of system components. Specifically, for GSS, we constructed 18 cases by adjusting the numbers of available pumps (m =1 to 3) and customers (n = 1 to 6); for CSS and CSSr, we constructed 9 cases by increasing the number of clients gradually (k = 2 to 10); and for SHS, we followed the actual system design, and did not increase its size by adding extra components.

The results based on RaPiD and RaPiDr are compared in Table 1. The degree of abstraction can affect reduction strength, we therefore evaluate two extreme cases in RaPiDr: one is the minimum abstraction that does not hide any event; the other is the maximum abstraction that hides all events except those related to reliability specification. The default algorithm in RaPiDr is to hide all events except specification events in the initial round and refine from there if the result is not satisfactory. Thus, we can often achieve a reduction closer to the maximum one and the minimum abstraction marks the 'worst' case of our method. The (*para.*) column contains the parameters for different systems, i.e., the numbers of pumps and customers for GSS and clients for CSS and CSSr. States number and time cost for RaPiD without reductions are shown by exact values. To compare the effect of reduction, the results for RaPiDr are shown in terms of ratios in the table, i.e., the ratios of the results from RaPiD to that of RaPiDr. A higher ratio indicates a higher degree of reduction. In the case that RaPiD runs out of memory (OM) and RaPiDr does not, then the ratio is presented as infinity (∞).

We have three main observations from Table 1. First, there is a general trend that the efficiency of reduction in state space and time increases as the system becomes larger. Moreover, there are some cases, e.g., GSS(2, 6), GSS(3, 4) and CSSr(10), that RaPiD runs out of memory and RaPiDr can still handle with relatively small number of states generated. Second, we observe that the time is reduced considerably in RaPiDr. This is because many states and redundant loops are removed in RaPiDr, and the convergence rate based on the value iteration is improved. The reduction in the total time implies that the time cost for the reduction is much less than its savings. Third, different levels of abstractions can save the state space and time to different extents because the more events are hidden, the greater reduction can be achieved. We also find that without hiding any events, our reduction method can still reduce the state spaces for GSS, CSSr and SHS, this is because our method can also remove internal τ transitions.

In addition, we notice that RaPiDr provides significantly better reductions on CSSr compared to the ones on CSS. Because CSS has few internal events, the state space is not reduced via the minimum abstraction, thus more time is spent. In contrast, the failure-repair loops in individual components of CSSr result in a dramatically large global space in RaPiD. However, they are effectively removed in RaPiDr even via the minimum abstraction. Therefore, the number of states and the time are reduced significantly. We remark that, by hiding communication events, we can still achieve a reduction ratio of more than 5 in some CSS cases. Moreover, it should be noticed that accuracy of abstraction is not affected by the difference between $\mathcal{R}_{min}$ and $\mathcal{R}_{max}$, as demonstrated by CSS cases.

The comparisons on PRISM and RaPiDr are presented in Fig. 6. Exactly the same models are taken as inputs to PRISM and RaPiDr. Similarly, we show the best/worst possible reduction in RaPiDr. In Fig. 6 (a), x- and y-axis of the plot are the number of states generated by PRISM and RaPiDr, respectively. Thus, the region below the diagonal line (i.e., slope = 1) indicates fewer states are generated by RaPiDr than by PRISM. The lower the point is, the higher degree of reduction RaPiDr provides. Note that the number of states is plotted in a logarithmic scale to focus on the comparison in terms of the order of magnitude. 'OM' stands for 'out of memory' and is only indicated qualitatively in the plot. We observe that RaPiDr is much more scalable than PRISM as all the scatters are within the lower-half region,

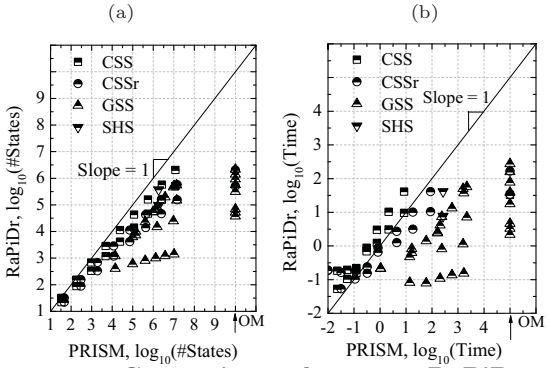

Figure 6: Comparisons between RaPiDr and PRISM on (a) states and (b) time (unit: second) in logarithmic scale

and it can reduce the number of states by several orders of magnitude. There are several cases that PRISM runs out of memory whereas RaPiDr can still keep the number of states within 10^7, evidenced by the points in the vertical 'OM' line.

Similarly, Fig. 6 (b) is a plot of the total time cost. We can observe that PRISM outperforms RaPiDr when the time taken is less than 10 seconds, although time factor is less critical for small systems. However, the benefit of adopting RaPiDr increases tremendously as the system grows larger. As shown in the plots, the scattered points tend to reside below and shift further away from the diagonal line for systems with larger states and longer verification time.

In summary, via comparing RaPiDr with RaPiD and PRISM, we find that our reduction method is more scalable and can reduce the total time significantly in many cases, especially for moderate and large systems. As RaPiDr reduces states and transitions in individual components, the overall states are reduced exponentially. Together with the removal of loops, reliability assessment can be speeded up significantly.

RQ 2: How accurate is the assessment using our technique?

This question is essentially related to the validation of our assumption that ignoring certain events does not change the assessment result significantly. First, we compare assessment results from RaPiDr based on the maximum (hereafter approximations) and those based on minimum abstraction (hereafter actual results). The maximum abstraction hides all communication events except specification events, thus it just provides approximations; and the minimum abstraction does not hide any communication events, thus it provides as accurate results as RaPiD does. The results are presented in Table 1, where $\mathcal{R}_{min}, \mathcal{R}_{max}$ are the actual results and $\mathcal{R}'_{min}$, $\mathcal{R}'_{max}$ are the approximations based on the maximum abstraction. The reliability results are reported to an accuracy up to five decimal places. It should be noted that the accuracy of prediction (i.e. difference between the actual result and approximation) is dominated by the choice of visible events. Further increasing the number of decimal places during computations can improve the precision but has been found to introduce insignificant effect on the accuracy. In Table 1, $\mathcal{R}'_{min}$ is always less than or equal to $\mathcal{R}_{min}$, and $\mathcal{R}'_{max}$ is always larger than or equal to $\mathcal{R}_{max}$, which complies with our proof in Section 4.3. Moreover, we can achieve an approximation of the maximum reliability that is equal to the actual maximum for all these cases.

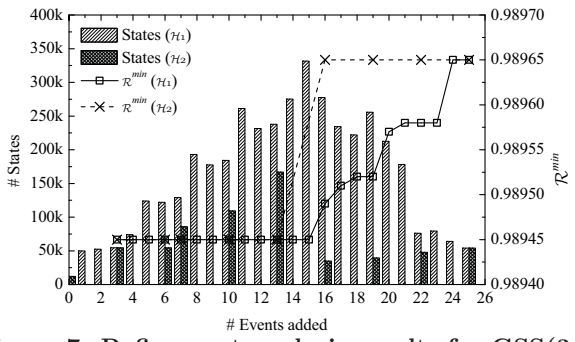

Figure 7: Refinement analysis results for GSS(3, 1)

Comparing the minimum reliabilities ($\mathcal{R}_{min}$ and $\mathcal{R}'_{min}$), we can observe that: (1) for CSS and CSSr, results are not affected by the abstraction, and thus we can achieve the maximum reduction. (2) for GSS system, the results are accurate except for some cases when m is less than n. This is because the details for the customers and queues have been hidden after abstraction. For example of GSS(3, 1), the discrepancy between the approximation (0.98945) and the actual result (0.98965) is 0.2%. If the reliability requirement is to ensure a reliability above 0.989, the result is conclusive and the verification can stop with the maximum reduction; otherwise, the refinement process will be carried out. (3) for the SHS system, the minimum reliability calculated is 0. This means there is no useful information obtained in the first round of abstraction and reduction, and some more events should be added back in the subsequent refinement step. In the following, we experimentally show that actual results can be obtained after several rounds of refinements.

We refine the abstract model by adding communication events back to improve the approximations. To have a quantitative evaluation, we keep track of the number of states and the assessment results during iterations of abstraction-refinement, and discuss the effect of the two refinement heuristics. In order to obtain a complete picture on how the state space and the assessment result change through refinements, instead of verifying against a given reliability requirement in which case our approach may terminate early, we continue until all of the communication events have been added into Σ_v. In the following part, we take GSS(3, 1) for an example.

There are 25 communication events for GSS(3, 1), apart from the events in the specification. We compare the above-mentioned two heuristics in each refinement step, and the results are shown in Fig. 7. The numbers of states are described in two bar charts and values can be read off via the y-axis on the left; and the resulting minimum probabilities are described in two sets of marked scatters and their values can be read off via the y-axis on the right. According to $\mathcal{H}_1$, 25 events are ranked according to the number of components synchronizing on the event. During each refinement, Σ_v is enlarged by adding one new event. Therefore, there are 25 bars and 25 points for $\mathcal{H}_1$. The event selection completes within one step and the time cost is negligible. For $\mathcal{H}_2$, we set the weighting factor w to 100 and divide the events into 10 groups according to the symmetry. We add the events by groups into Σ_v so there are 10 bars and points for $\mathcal{H}_2$ in Fig. 7. The total time spent on selecting events based on $\mathcal{H}_2$ is 610 seconds.

We have the following observations. First, the calculated minimum reliabilities converge to the actual result, regardless of the heuristics. Second, by comparing two heuristics,

Table 2: Reliability assessment against reliability requirements

requirement		GSS(3, 1)	CSS(8)	CSSr(8)	SHS
$\mathcal{R}^{min} \geq 0.40$	result	valid	valid	valid	valid
	time (s)	10.06	3.18	3.22	3,039
	#iteration	4	1	1	34
$\mathcal{R}^{min} \geq 0.90$	result	valid	invalid	valid	invalid
	time (s)	12.14	100.00	3.00	3,037
	#iteration	4	19	1	34
$\mathcal{R}^{min} \geq 0.99$	result	invalid	invalid	invalid	invalid
	time (s)	649.48	98.99	95.58	3,040
	#iteration	26	19	19	34
$\mathcal{R}^{min}$	result	0.98965	0.43047	0.91540	0.63274
$\mathcal{R}^{max}$	result	1.00000	1.00000	1.0000	1.00000

we can find $\mathcal{H}_2$ outperforms $\mathcal{H}_1$. The number of states generated by $\mathcal{H}_2$ is much less than that by $\mathcal{H}_1$; and the resulting probability converges to the actual result (0.98965) much faster for $\mathcal{H}_2$. At a particular point based on $\mathcal{H}_2$, when 16 events are added, the approximation reaches the actual result while the state space remains relatively small. This information can be used to guide reliability assessment for even larger GSS systems, i.e., the similar set of events can always be selected first. We remark that although $\mathcal{H}_2$ can provide relatively better performance than $\mathcal{H}_1$, it requires extra knowledge in dividing events into groups and more efforts in evaluating the effectiveness of each candidate group at each refinement iteration. Last, although the size of the state space may increase after refinement, it remains manageable along the way, i.e., much less than that generated by RaPiD (without the reductions) and PRISM.

Similar refinements have been conducted on SHS. As a result, we have identified a group of 24 (out of 33) events based on which the approximated minimum reliability converges to the actual result. The number of states is $73,054$ and verification time is 9.4s.

In summary, starting from hiding all events not in the specification, we can obtain safe approximations on the verification result. Refinement can incrementally help to improve accuracy. Comparing two heuristics, the results show that $\mathcal{H}_2$ often guides the refinement better than $\mathcal{H}_1$ while time cost of using $\mathcal{H}_1$ is much lower.

RQ 3: How efficient is our method given a requirement?

The efficiency of our approach is directly related to the reliability requirement. Intuitively, our approach terminates quicker given a less restrictive requirement. The question is then how sensitive our approach is regarding different reliability requirements. Table 2 shows the reliability assessment against different levels of reliability requirements, i.e., 0.40, 0.90, and 0.99. If the system's minimum reliability is above or equal to a requirement, RaPiDr is expected to report 'valid'; and otherwise 'invalid'. Besides the assessment results, we also record the time and the number of abstraction/refinement iterations needed. If 'invalid' is reported, RaPiDr still reports the actual assessment results, i.e., $\mathcal{R}^{min}$ and $\mathcal{R}^{max}$, shown in the last two rows of the table. As shown, if the reliability requirement is 0.40, all results are 'valid' and RaPiDr terminates quickly with only a few iterations. If the requirement is $\mathcal{R}^{min} \geq 0.90$, results from CSS(8) and SHS become 'invalid' (as expected) and more time and more iterations are needed. If the requirement is $\mathcal{R}^{min} \geq 0.99$, all results become 'invalid'. We remark that, the refinement here is based on $\mathcal{H}_1$, which is fully automatic without any domain information. If there is some information to guide the refinement, fewer iterations can be expected.

We conclude that RaPiDr can terminate early when the requirement is low or the system is relatively reliable. For a less reliable system and relatively high requirement, RaPiDr may indeed take more iterations and time. In the worst case, it will run all the model versions between the maximum abstraction and minimum abstraction, thus take more time than that required to verify the minimum abstraction. This is similar to other approaches on alleviating the state explosion problem [20, 18, 40, 38, 36]. In practice, distributed systems can easily have many components such that the state space is too large and reliability assessment based on the concrete model is impossible. RaPiDr can always deliver a safe reliability approximation based on abstract models; and with more time and computing resources, RaPiDr can produce increasingly more accurate approximations.

6. RELATED WORK AND CONCLUSION

We proposed an abstraction and refinement framework to improve the performance of reliability assessment for distributed systems by controlling communication events among distributed components. We also proposed two heuristics to guide refinement step automatically. Our empirical studies showed that our method could reduce the state space by several orders of magnitude and speed up the assessment.

In reliability assessment, some works have considered each component as a single node and composed all the nodes in one Markov chain according to the system architectures or service usage scenarios [10, 37, 30, 28, 21, 7, 15, 27, 52, 25]. Some works have modeled multi-agent systems as Markov chains [16] or MDPs [51] and verified over LTL properties. In our work, we perform reliability analysis for distributed systems based on a set of MDPs. We further apply abstraction and refinement on the communication events to reduce state spaces.

There are two main approaches on alleviating state space explosion via the compositional verification. One is counterexample guided abstraction refinement (CEGAR) [11, 12], and its extension in probabilistic system [36]. [36] works on predicates abstraction and refinement and only calculates upper bounds on maximum probabilities. Our approach works on communication events among components, and can provide both lower and upper bounds without enumerating possibly large or even infinite set of paths. The other approach is assume-guarantee verification [44, 35, 26], with its extensions in probabilistic systems [20, 18, 40, 38, 24, 45, 8, 33, 4, 3, 38]. The challenge is on automatically generating small assumptions. In many assume-guarantee reasoning works, there is no guidance on how the system shall be decomposed to achieve good performance [14]. In contrast, our method works on MDPs and reduces every component according to a subset of the communication events, thus there is no need to find assumptions or suitable decomposition. In fact, our approach is orthogonal to those compositional verification approaches. It can be used prior to those compositional verification approaches for even larger systems that no single method can handle alone; i.e., the individual component is first reduced via the attraction and reduction steps and the compositional approaches are then applied to perform verification on the resulting components.

7. REFERENCES

[1] http://www.comp.nus.edu.sg/~pat/rel/distributed.

[2] RaPiD. http://www.comp.nus.edu.sg/~pat/rapid.

[3] E. Ábrahám, N. Jansen, R. Wimmer, J.-P. Katoen, and B. Becker. DTMC model checking by SCC reduction. In *International Conference on Quantitative Evaluation of SysTems*, pages 37–46, 2010.

[4] M. E. Andrés, P. R. D'Argenio, and P. van Rossum. Significant diagnostic counterexamples in probabilistic model checking. In *Haifa Verification Conference*, pages 129–148, 2008.

[5] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[6] C. Baier and J. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[7] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, Sept. 2012.

[8] S. Chaki and O. Strichman. Optimized l*-based assume-guarantee reasoning. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 276–291. Springer, 2007.

[9] K. Chatterjee and T. A. Henzinger. Value iteration. In *25 Years of Model Checking*, pages 107–138. Springer, 2008.

[10] R. C. Cheung. A user-oriented software reliability model. *IEEE Trans. Software Engineering*, SE-6(2):118–125, 1980.

[11] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.

[12] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[13] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[14] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: an investigation of decomposition for assume-guarantee reasoning. In *International Symposium on Software Testing and Analysis*, pages 97–108. ACM, 2006.

[15] V. Cortellessa and V. Grassi. Reliability modeling and analysis of service-oriented architectures. In *Test and analysis of web services*, pages 339–362. Springer, 2007.

[16] M. I. Dekhtyar, A. J. Dikovsky, and M. K. Valiev. Temporal verification of probabilistic multi-agent systems. In *Pillars of computer science*, pages 256–265. Springer, 2008.

[17] C. Eisentraut, H. Hermanns, J. Schuster, A. Turrini, and L. Zhang. The quest for minimal quotients for probabilistic automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 16–31. Springer, 2013.

[18] L. Feng, T. Han, M. Kwiatkowska, and D. Parker. Learning-based compositional verification for synchronous probabilistic systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 511–521. Springer, 2011.

[19] L. Feng, M. Kwiatkowska, and D. Parker. Compositional verification of probabilistic systems using learning. In *International Conference on Quantitative Evaluation of SysTems*, pages 133–142. IEEE CS Press, 2010.

[20] L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In *International Conference on Fundamental Approaches to Software Engineering*, pages 2–17. Springer, 2011.

[21] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *International Conference on Software Engineering*, pages 341–350. ACM, 2011.

[22] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *Formal Methods for Eternal Networked Software Systems*, Springer, 2011.

[23] M. Gheorghiu, D. Giannakopoulou, and C. S. Păsăreanu. Refining interface alphabets for compositional verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 292–307. Springer, 2007.

[24] M. Gheorghiu, C. S. Păsăreanu, and D. Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *International Conference on Computer Aided Verification*, pages 135–148. Springer, 2008.

[25] C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburrelli. Mining behavior models from user-intensive web applications. In *International Conference on Software Engineering*, pages 277–287, 2014.

[26] D. Giannakopoulou, C. S. Păsăreanu, and J. M. Cobleigh. Assume-guarantee verification of source code with design-level assumptions. In *Proceedings of the 26th international conference on software engineering*, pages 211–220. IEEE Computer Society, 2004.

[27] S. Gilmore, L. Gönczy, N. Koch, P. Mayer, M. Tribastone, and D. Varró. Non-functional properties in the model-driven development of service-oriented systems. *Software and System Modeling*, 10(3):287–311, 2011.

[28] S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Trans. Dependable and Secure Computing*, 4(1):32–40, 2007.

[29] G. Gössler and J. Sifakis. Component-based construction of deadlock-free systems: Extended abstract. In *Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 420–433. Springer, 2003.

[30] K. Goševa-Popstojanova and K. S. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2-3):179–204, 2001.

[31] L. Gui, J. Sun, Y. Liu, Y. J. Si, J. S. Dong, and X. Y. Wang. Combining model checking and testing with an application to reliability prediction and distribution. In *International Symposium on Software Testing and Analysis*, pages 101–111. ACM, 2013.

[32] L. Gui, J. Sun, S. Song, Y. Liu, and J. S. Dong. SCC-based improved reachability analysis for markov decision processes. In *International Conference on Formal Engineering Methods (ICFEM)*. Springer, 2014.

[33] T. Han and J.-P. Katoen. Counterexamples in probabilistic model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 72–86. Springer Berlin Heidelberg, 2007.

[34] D. Heimbold and D. Luckham. Debugging ada tasking programs. *EEE Software*, 2(2):47–57, 1985.

[35] T. A. Henzinger, S. Qadeer, and S. K. Rajamani. You assume, we guarantee: Methodology and case studies. In *Computer Aided Verification*, pages 440–451. Springer, 1998.

[36] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic cegar. In *International Conference on Computer Aided Verification*, pages 162–175. Springer, 2008.

[37] A. Immonen and E. Niemel. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and Systems Modeling*, 7(1):49–65, 2008.

[38] A. Komuravelli, C. S. Păsăreanu, and E. M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In *International Conference on Computer Aided Verification*, pages 310–326. Springer, 2012.

[39] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification*, pages 585–591, 2011.

[40] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 23–37. Springer, 2010.

[41] J. C. Laprie and K. Kanoun. *Handbook of software Reliability Enginerring*, chapter Software Reliability and System Reliability, pages 27–69. McGraw-Hill, New York, NY, 1996.

[42] Y. Liu, L. Gui, and Y. Liu. Mdp-based reliability analysis of an ambient assisted living system. In *Internation Symposium on Formal Methods Industry Track*, Singapore, May 2014.

[43] Y. Liu, X. Zhang, J. S. Dong, Y. Liu, J. Sun, J. Biswas, and M. Mokhtari. Formal analysis of pervasive computing systems. In *International Conference on Engineering of Complex Computer Systems*, pages 169–178, 2012.

[44] C. S. Păsăreanu, M. B. Dwyer, and M. Huth. Assume-guarantee model checking of software: A comparative case study. In *Theoretical and Practical Aspects of SPIN Model Checking*, pages 168–183. Springer, 1999.

[45] C. S. Păsăreanu, D. Giannakopoulou, M. Bobaru, J. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008.

[46] A. Pnueli. The Temporal Logic of Programs. In *Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[47] A. W. Roscoe. Model-checking CSP. *A classical mind: essays in honour of CAR Hoare*, pages 353–378, 1994.

[48] S. Song, L. Gui, J. Sun, Y. Liu, and J. S. Dong. Improved reachability analysis in DTMC via divide and conquer. In *International Conference on Integrated Formal Methods*, pages 162–176, 2013.

[49] J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *International Conference on Computer Aided Verification*, volume 5643 of *LNCS*, pages 709–714. Springer, 2009.

[50] J. Sun, S. Z. Song, and Y. Liu. Model checking hierarchical probabilistic systems. In *International Conference on Formal Engineering Methods*, pages 388–403, 2010.

[51] M. Valiev and M. Dekhtyar. Complexity of verification of nondeterministic probabilistic multiagent systems. *Automatic Control and Computer Sciences*, 45(7):390–396, 2011.

[52] W.-L. Wang, D. Pan, and M.-H. Chen. Architecture-based software reliability modeling. *Journal of Systems and Software*, 79(1):132–146, 2006.