11-2014

RaPiD: A toolkit for reliability analysis of non-deterministic systems

Lin GUI

Jun SUN

Yang LIU

Truong Khanh NGUYEN

Jin Song Dong DONG

RaPiD: A Toolkit for Reliability Analysis of Non-deterministic Systems

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

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

ABSTRACT

Non-determinism in concurrent or distributed software systems (i.e., various possible execution orders among different distributed components) presents new challenges to the existing reliability analysis methods based on Markov chains. In this work, we present a toolkit RaPiD for the reliability analysis of non-deterministic systems. Taking Markov decision process as reliability model, RaPiD can help in the analysis of three fundamental and rewarding aspects regarding software reliability. First, to have reliability assurance on a system, RaPiD can synthesize the overall system reliability given the reliability values of system components. Second, given a requirement on the overall system reliability, RaPiD can distribute the reliability requirement to each component. Lastly, RaPiD can identify the component that affects the system reliability most significantly. RaPiD has been applied to analyze several real-world systems including a financial stock trading system, a proton therapy control system and an ambient assisted living room system. The is available at `http://fse22.gatech.edu/cfp/demos`

Categories and Subject Descriptors

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

Keywords

reliability analysis, Markov Decision Process, non-determinism

1. INTRODUCTION

Nowadays, virtually any industry, e.g., automotive, avionics, oil, telecommunications and banking, is highly dependent on computer systems for their automated functioning. Failures would damage the reputation of the system operators, and potentially lead to losses in capitals or even human lives. To prevent those losses, performing reliability analysis on the system before its deployment is highly desirable. Thus, tools for assisting reliability analysis are in an overwhelming need to help project managers in project planning such as risk mitigation, resource allocation in testing phase, and strategy appraisal.

Existing approaches on reliability analysis, together with the associated tools, fall into two categories: black-box approaches [7] and white-box approaches [3, 9]. The black-box approaches treat a system as a monolith and evaluate its reliability using testing techniques. They use the observed failure information to predict the reliability of software based on several mathematical models. On the contrary, the white-box approaches assume that the reliability of each system component is known and evaluate software reliability analytically based on the model of the system architecture. Typical models include discrete or continuous time Markov chains (DTMCs, CTMCs) [3, 9]. Those approaches assume the system is deterministic, i.e., given the same inputs, the outputs of the system are always the same. Thus, in their reliability models, the transition probabilities among components are assumed to be known. All those approaches assume that there is only one probability distribution for the possible usage of a component.

However, as software becomes more complex and often operates in a distributed or dynamic environment, the execution orders among software components are hard to measure prior to the software deployment (i.e., exhibiting non-deterministic behaviors). For example of a system with two servers running in a distributed environment, the execution frequency of each server depends on the specific runtime task that is required from external. Without priori knowledge on tasks, no statement can be made about the likelihood with which sever is selected. In fact, non-determinism also exists in many other modern software, e.g., a cloud computing system within which multiple processes aim to access shared resources and a pervasive system within which the software intensively interacts with environments or human behaviors.

In order to cope with those non-deterministic behaviors, we propose to perform reliability analysis based on a system model in the form of a Markov decision process (MDP), a popular formalism to model both probabilistic and non-deterministic behaviors [5]. Towards this goal, in this work, we present a toolkit called RaPiD (Reli*a*bility Pred*i*ction and Distribution) to assist automatic reliability analysis. Using probabilistic model checking techniques, e.g., (parametric) reachability checking, three fundamental and highly important reliability issues can be investigated in RaPiD for non-deterministic systems: (1) RaPiD can synthesize the overall system reliability given the reliability values of system components; (2) given a reliability requirement on the overall system, RaPiD can distribute the reliability requirement to each component; and (3) RaPiD can identify the component that affects the system reliability most significantly. To further enhance the efficiency and scalability of reliability analysis, RaPiD incorporates various techniques, i.e., strongly connected components elimination to solve the slow convergence issue in probability computation, and abstraction and refinement on the communications between distributed systems to alleviate the state explosion issue. For its applications, RaPiD has been used for the reliability analysis of several real-world systems.

FSE'14, November 16–21, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3056-5/14/11...$15.00
http://dx.doi.org/10.1145/2635868.2661668

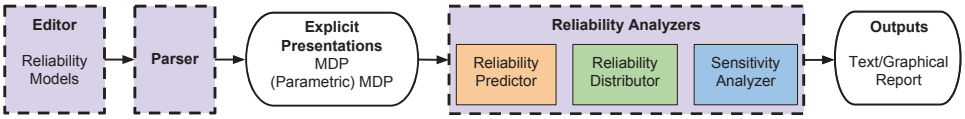

Figure 1: RaPiD architecture

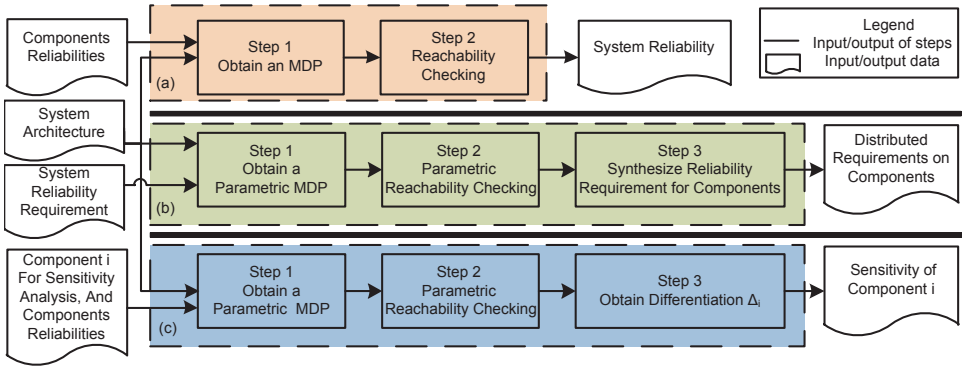

Figure 2: RaPiD overview: (a) reliability prediction; (b) reliability distribution; (c) sensitivity analysis

2. TOOL OVERVIEW

RaPiD is a self-contained reliability analysis toolkit, which consists of three main components, i.e., Editor, Parser, and Reliability Analyzers. Fig. 1 shows the architecture of RaPiD. In the graphical editor, a system model for reliability analysis (i.e., reliability model) is first created, from which the explicit model, i.e., MDP, is then automatically obtained by the parser. The core algorithms for reliability analysis are in the reliability analyzers including reliability predictor, reliability distributor and sensitivity analyzer. After the analysis, RaPiD presents results in a text report or a graphical plot. In the following, we introduce the reliability model and then present the workflow for the three reliability analyzers.

2.1 Reliability Model

Extended from Cheung's model [3], the model for reliability analysis in our setting is an MDP $\mathcal{M}$ that can be built from system architecture and user environments. It can support the modeling of both probabilistic and non-deterministic behaviors. In this model, states and transitions are two key elements constructed as follows.

States Each system component C is a self-contained piece of codes that can be independently designed, implemented, and tested. Each system component represents a state in MDP. In addition, there are two absorbing states: a state of *Success* and a state of *Failure*. A simple model is demonstrated in Fig. 3. For compact reliability model presentation, we skip the *Failure* state. Instead, a node labelled as $C(R_c)$ is used to denote a component C with a probability of R_c to transit to the successive components, and a probability of $1 - R$ to reach *Failure* state.

Transitions The transition probability in a probability distribution at each edge represents the usage information, e.g., P_{ij}, is the probability from component i to component j, given that component i does not fail. An MDP can have more than one probability distributions at a state. This feature enables RaPiD to model all possible operating environments/situations explicitly. RaPiD supports the modeling of failure handling mechanism in the system, where the transition with probability $1 - R_C$ leads to a failure recovering state instead of the $Failure$ state. Taking $Server_1$ in Fig. 3 as an example, its reliability can be read off from the graph as 0.9972 and it has two outgoing transitions labeled with action η. If $Server_1$ terminates successfully, it has a probability 0.584 of going to $Exit$, and a probability 0.416 of going to database DB. If $Server_1$ fails, it goes to $Server_2$, which serves as a backup server for $Server_1$. This backup transition is denoted by the dash line in the figure.

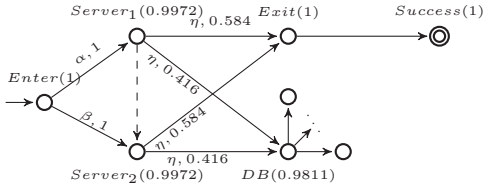

Figure 3: A reliability model

2.2 Reliability Analysis

With different input knowledge and requirements, RaPiD can readily address three different questions on software reliability.

- "What is the overall system reliability if the reliability of each component is known, considering all possible user behaviors, and unreliable factors?"

This is the problem of ***reliability prediction***. This question is to be answered necessarily before system deployment since end users would prefer to know how reliable the system is. The reliability value of each component and an MDP model of system are required for predicting the overall system reliability. Reliability prediction is equivalent to checking the probability that the system never fails. It is then transformed into a problem of calculating the probability of reaching accepting nodes from an initial state to a goal state s on an MDP model $\mathcal{M}$, denoted as $Pr(\mathcal{M}, s)$. Here, RaPiD performs value iteration approach [2] to compute the reachability probabilities. Unlike DTMC approaches, the result here is a probability range due to the non-deterministic behaviors. The upper bound is the system reliability corresponding to the best scenario in the systems, whereas, the lower bound corresponds to the worst scenario.

- "What is the reliability required on a certain component if there is a requirement on overall system reliability?"

This is the problem of ***reliability distribution***. Addressing this issue is useful because we can have specific quantitative requirements on the selection of software and hardware components, whose qualities are often cost-sensitive. The reliability distribution analysis shown in Fig. 2(b) needs two inputs: (1) a reliability requirement R on the overall system; (2) a parametric MDP model $\mathcal{M}$. RaPiD considers only memoryless schedulers that have already been proven

to be enough for probability reachability analysis in MDPs [2]. Given a scheduler δ, we can obtain the system reliability (i.e., $Pr(\mathcal{M}_\sigma, s)$) as a polynomial function of x and its associated inequality, e.g., $0.5x^1 + 0.16435x^3 + 0.05402x^5 + \cdots \geq R$. To solve the constraints on an individual component, RaPiD uses Newton's method, due to its fast convergence rate to the solution/root. RaPiD calculates the lower bounds on x for finitely many schedulers among which the maximum value gives us the minimum requirement on component reliability.

- "Which component is most critical to system reliability among all system components?"

The answer can be addressed via ***sensitivity analysis***. This analysis is essential to improve the overall system reliability effectively with limited resource. For example, if a system is shown to be not reliable enough based on current components, it is desirable to prioritize the components such that reliability improvement of a higher priority component would result in better improvement on overall system reliability. Sensitivity analysis requires all component reliabilities to be known in advance and an indication on which one of those components needs to be evaluated, as shown in Fig. 2 (c). The sensitivity Δ_i of component i with reliability R_i is defined as a partial derivation of system reliability R, i.e., $\Delta_i = \frac{Pr(\mathcal{M},s)}{\delta R_i}$. Here, $Pr(\mathcal{M}, s)$ are polynomials obtained via reliability distribution. RaPiD has equipped with polynomial solvers to solve these differential equations.

2.3 Enhancements

We have developed two techniques to enhance the efficiency and scalability in reliability analysis.

Efficiency We improve the efficiency of reachability checking, which is fundamental in our reliability analysis. It is known that existing approaches on reachability analysis for Markov models are often inefficient (i.e., slow convergence) when a given model contains a large number of states and loops. In RaPiD, we implemented strongly connected components (SCCs) elimination methods, and actively removed redundant probability distributions by finding convex hull [11, 6].

Scalability We improve the scalability of reliability prediction, in particular, for distributed systems. RaPiD can support the modeling of parallel composition of a set of MDPs and its associated refinement checking. Our method relies on probabilistic model checking techniques, which is limited to small scale distributed systems. In RaPiD, we improve the probabilistic model checking through abstraction and reduction, which control the communications among different distributed local systems and actively reduce the size of each system.

2.4 Assumptions and Threads to Validity

This work shares the same assumptions with the conventional component-based reliability analysis [4]. It assumes that there is statistical independence among failures of the components. Moreover, in our reliability analysis, we assume that any component's failure will eventually result in the failure of the system. Self-recovery/repair scenarios are not considered as failure cases, and these scenarios can all be modeled in RaPiD.

3. IMPLEMENTATION AND AVAILABILITY

RaPiD is implemented to provide a friendly user interface to draw reliability models and fully automated methods to the reliability analysis (i.e., reliability prediction, reliability distribution, and sensitivity analysis). It has approximately 6.5K lines of C# code. It uses a number of MATLAB (version 2009a) libraries to support powerful mathematical calculations, as well as graph plotting functions. Starting from 2012, RaPiD has come to a stable stage with solid testing and has been applied to analyze several real-world systems. The tool (along with a screencast illustrating its usage) is available at [1].

Table 1: Reliability prediction for the three models

Name (#Schedulers)	M1 (1)	M2 (5)	M3 (160)
Min. Reliability	0.95568	0.95401	0.73149
Max. Reliability	0.95568	0.95568	0.96257

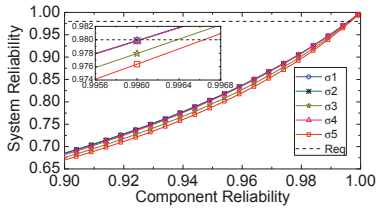

Figure 4: Reliability distribution results for M2b

4. EVALUATION

4.1 Practical Case Studies

So far, RaPiD has been successfully used in reliability analysis of three real-world systems. Due to space limitation, we only present a small part of evaluation results for demonstration. The detailed descriptions and comprehensive evaluation results are available at [1].

Call Cross System (CCS) It is from our industrial collaborator, a financial software solution provider. CCS is a stock trading system accepting order flow in a global operating environment. Using RaPiD, six different models of CCS system that vary only on the 'uncertainty' degree of the deployment environment (i.e., M1, M2 and M3) and the existence of back-up servers (i.e., M1b, M2b and M3b), are evaluated via reliability prediction and reliability distribution. As the reliability of a system changes with the dynamic operating environment, reliability prediction results in RaPiD are the maximum and minimum reliabilities as listed in Table 1. From the table, we can find that the difference between maximum and minimum reliabilities becomes larger when the number of non-deterministic choices (i.e., 'uncertainty') increases from M1 to M3. For reliability distribution, the reliability requirement is generated on each component, as well as the plots of system reliability versus component reliability as shown in Fig. 4 for model M2b. Reliability requirement is shown as the dashed horizontal line and distributed reliability requirements are determined at the intersection points. A comparison of the six models is shown in Fig. 5. We observe that the system reliability indeed becomes higher by introducing a backup server, but with the increase of component reliability, the gain of system reliability by introducing the backup server decreases. Readers are referred to [5] for more details.

Therapy Control System (TCS) It is from the Burr Proton Therapy Center that provides a radiation therapy facility associated with a hospital in Boston. Given the requirement on system level reliability, it is desirable to generate concrete reliability requirements for newly developed components so that they are contracted properly. The reliability analysis was conducted during an upgrading of TCS. The challenge in applying RaPiD was that there was no precise information on transition probabilities. As a result, transitions in the system have been modeled as non-deterministic transitions only. The results are shown in Fig. 6. We have shown that RaPiD can still obtain some useful results [5]. For example, although there

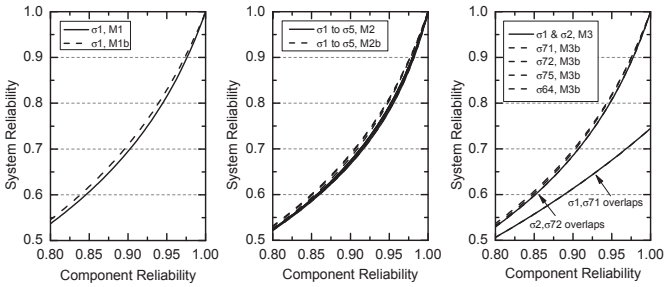

Figure 5: System reliability vs. component reliability for CSS models: (a)M1 and M1b; (b)M2 and M2b; (c)M3 and M3b

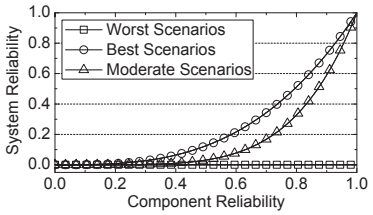

Figure 6: Reliability analysis result for TCS

are 2,592 schedulers, only three different system reliabilities (in terms of polynomials of the component reliability) exist. By further analyzing the corresponding schedulers reported by RaPiD, we can identify three typical workflows of the system that result in the three scenarios, respectively.

Ambient Assisted Living System (AAL) It is a smart healthcare system named AMUPADH for elderly people with dementia in a nursing home in Singapore. This system is highly interactive that it can automatically react according to users' behaviors. With 6 months trial deployment, the system turned to be highly unreliable and fail frequently. Thus, reliability analysis has been carried out with RaPiD on different usage scenarios. Reliability prediction results are presented in Table 2, which show that the overall system reliability can hardly reach 0.5. Given system reliability requirement, the distributed requirement on sensors' reliabilities are listed in Table 3. From the sensitivity analysis, we have identified that the overal system reliability can be improved most effectively by improving the Wi-Fi network. Details are referred to an industrial case study report in [10].

4.2 Performance and Scalability

RaPiD is efficient in our case studies. The reliability prediction took 0.03 seconds for the CCS, and the reliability distribution took 42 seconds for the CCS (with 160 schedulers) and 628 seconds for the TCS (with 2,592 schedulers). To further test the scalability of RaPiD, we evaluate RaPiD's reliability prediction and distribution using 5 benchmark MDP models as well as randomly generated models (with 1K to 50K states and the number of states for having multiple transitions are sampled from a uniform distribution). The results show that RaPiD can handle 14K states per second on average (with termination threshold defined by a relative difference of 1.0E-6) in calculating reachability probability. Reliability distribution (with a bound 600 on the number of terms in the obtained polynomial) is slightly slower due to maintaining/updating/solving the polynomial functions. All above data are obtained using a PC with Intel® Core(TM) i7 CPU at 2.80 GHz and 8 GB of RAM.

5. CONCLUSION

This paper presents a toolkit RaPiD, which is a self-contained toolkit for reliability analysis. Unlike other reliability analysis toolkits [8, 12] that only work for deterministic systems, RaPiD can work for the analysis of non-deterministic systems that may be exposed to various possible operating environments. Although RaPiD uses probabilistic model checking techniques, unlike general probabilistic model checkers, RaPiD is tailored to software reliability analysis. In particular, it provides fully automated solutions to reliability prediction and reliability distribution problems as well as sensitivity analysis. RaPiD has been successfully applied for the reliability analysis of real-world systems such as a financial system, a therapy control system and a smart living room system.

Table 2: Results of reliability prediction for AAL

Rel.	UWB	SBTL	SNS	STL	TNO	WiW
#Sdl.	32	24	32	16	64	16
Max.	0.3744	0.4190	0.3670	0.3707	0.3707	0.3707
Min.	0.2956	0.2463	0.2897	0.2927	0.2897	0.2927
Time	<1 ms					

Sdl. = Schedulers; Rel. = Reliability

Table 3: Distributed reliability requirements on sensors

Req. R	UWB	SBTL	SNS	STL	TNO	WiW
0.4	0.854	0.904	0.913	0.911	0.911	0.911
	0.886	0.938	0.941	0.923	0.923	0.923
0.5	0.914	N.A.	0.965	0.963	0.963	0.963
	0.996	N.A.	0.995	0.994	0.994	0.994
Time (s)	3.45	2.68	3.86	1.87	11.00	2.35

6. REFERENCES

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

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

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

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

[5] 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 *ISSTA*, pages 101–111. ACM, 2013.

[6] L. Gui, J. Sun, S. Song, Y. Liu, and J. S. Dong. SCC-based improved reachability analysis for Markov decision processes. In ICFEM. Springer, 2014.

[7] 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.

[8] A. M. Johnson, Jr. and M. Malek. Survey of software tools for evaluating reliability, availability, and serviceability. *ACM Comput. Surv.*, 20(4):227–269, Dec. 1988.

[9] J. C. Laprie and K. Kanoun. *Handbook of software Reliability Enginering*, chapter Software Reliability and System Reliability, pages 27–69. McGraw-Hill, 1996.

[10] Y. Liu, L. Gui, and Y. Liu. Mdp-based reliability analysis of an ambient assisted living system. In *FM Industry Track*, pages 688–702, Singapore, May 2014.

[11] S. Song, L. Gui, J. Sun, Y. Liu, and J. S. Dong. Improved reachability analysis in DTMC via divide and conquer. In *IFM*, pages 162–176. Springer, 2013.

[12] W.-L. Wang, D. Pan, and M.-H. Chen. Architecture-based software reliability modeling. *J. Syst. Softw.*, 79(1), 2006.