SCIENCE CHINA
Information Sciences

• LETTER •

June 2018, Vol. 61 069101:1–069101:3
https://doi.org/10.1007/s11432-016-9220-5

Evaluation of redundancy-based system: a model checking approach

Ling FANG[1], Chunyan MU[2], Zhuo CHENG[3] & Guoqiang LI[4]*

[1]*Institute of Technology Innovation, Hefei Institutes of Physical Science, Chinese Academy of Sciences, Hefei* 230088*, China;*
[2]*School of Computing, Teesside University, Middlesbrough* TS1 3BX*, United Kingdom;*
[3]*State International S&T Cooperation Base of Networked Supporting Software, Jiangxi Normal University, Nanchang* 330022*, China;*
[4]*School of Software, Shanghai Jiao Tong University, Shanghai* 200240*, China*

Received 14 December 2016/Revised 10 June 2017/Accepted 24 July 2017/Published online 28 March 2018

Citation Fang L, Mu C Y, Cheng Z, et al. Evaluation of redundancy-based system: a model checking approach. Sci China Inf Sci, 2018, 61(6): 069101, https://doi.org/10.1007/s11432-016-9220-5

Dear editor,
Functional redundancy is a key and well-known fault tolerance scheme for which there are functionally equivalent versions that can rescue a system from execution failure and improve its reliability with standby. It is fundamental in the establishment of effective and formal evaluations of system reliability [1–3].

Probabilistic model checking is a method for the modeling and analysis of systems that exhibit random or probabilistic behaviors [4]. In classical model checking, a model of the probabilistic system is built and then subjected to algorithmic analysis to establish whether it satisfies a given specification.

In this article, we propose a method in which we model redundancy-based systems as discrete-time Markov chains and specify the properties of interest for analysis using probabilistic computational tree logic (PCTL). To satisfy reliability requirements, this method can guarantee the correctness of the redundancy strategies and provide accurate evaluations based on scientific evidence regarding the redundant system. In this article, we use the PRISM model checker [5] as a framework for modeling and analyzing systems. We introduce an evaluation method for redundancy-based systems via a running example, as shown in Figure 1.

Redundancy-based system.

• Conventional schemes. There are two common functional redundancy schemes: the recovery block (RB) and N-version programming (NVP), and their corresponding evaluation methods are relatively mature [1, 2].

Traditionally, users adopt approximate existing models and apply their evaluation based on several hypotheses, e.g., the multiple versions are independent, which may cause deviation in accuracy [6].

• Running example. Here, we consider a running example of a redundancy-based system that uses a reset signal (RST) and non-mask interrupt (NMI) [7] to periodically trigger the "save" and "reset" processes [8], respectively, and backup data is restored to the system if the current module returns false, as shown in Figure 1(a). This system is designed to resist electromagnetical interference, which can cause timing or transient faults such as the stop, freeze, and lockup of a device or piece of equipment. These failures are annoying, they occur frequently, and they are rarely diagnosed [9].

As shown in Figure 1(b), the redundant versions are a number of copies of data $D_1, D_2, \ldots, D_n$ that can be used to recover the application. $\mathrm{AT}_1, \ldots, \mathrm{AT}_n$ are responsible for checking the cor-

* Corresponding author (email: li.g@sjtu.edu.cn)

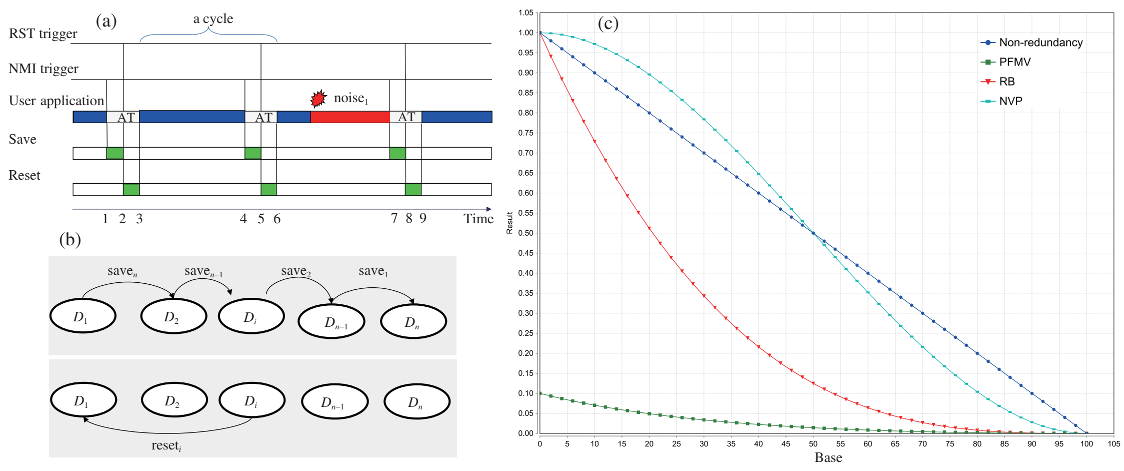

Figure 1 (Color online) Redundancy strategy and evaluation of a running example. (a) The typical process of recovery; (b) the copy process on D_1–D_n; (c) the values of formula (3) for RB, NVP, and PFMV

rectness of the data and creating backup and recovery copies. If AT$_1$ returns true, D_1 is stored, and a set of copies of data are backed up in the order $D_{i-1} \to D_i$ $(1 < i \leqslant n)$. If AT$_1$ returns false, $D_2, D_i, \ldots, D_n$ are checked in order until the valid data D_i is found, and D_1 is recovered from D_i. These processes are repeated periodically on the time-line. As such, D_i always holds the copy of D_1 when the system executes before i cycles.

• Evaluation issues. This running example is unique and not as easily approximated as RB or NVP. Its approximations and hypotheses deviate from typical modeling and analysis approaches. In addition, there are many properties that are desired by industry but impossible to verify, e.g., the probability that an endless freezing will be lower than a desired value.

Over approximation misleads users into devising too little redundancy for the same reliability requirements. In contrast, insufficient assessment misleads users into wasting resources by devising too much redundancy for the same reliability requirements.

Evaluation of redundancy-based software systems. Our method PFMV (probabilistic formal modelling and verification) evaluates redundancy-based software systems using probabilistic model checking.

Using the modeling language of PRISM [5], we can accurately describe the relations of multiple versions even when the strategy is unique, with no limits with respect to the type of existing models. Moreover, it does not require the assumption of the independent multiple versions.

When an accurate model is achieved, we evaluate the formal specifications to gain insight into the redundant strategy. Eqs. (1)–(6) present several typical analyses that are frequently used in industry [1], which we explain below.

$$\boldsymbol{A}[\boldsymbol{G}\ \mathrm{RST} \Rightarrow (P_{\geqslant 1}[\boldsymbol{X}(\mathrm{APP} \wedge \neg\mathrm{NMI} \wedge \neg\mathrm{RST})])], \tag{1}$$

$$\boldsymbol{A}[\boldsymbol{G}\ \mathrm{RST} \wedge (\neg D_1 \wedge \neg D_2 \wedge \neg D_3) \Rightarrow (P_{\geqslant 1}[\boldsymbol{X}(D_1 \wedge D_2 \wedge D_3)])], \tag{2}$$

$$\mathrm{PFD} = \mathrm{filter}(\mathrm{avg}, (S_{=?}[\text{“reboot”}]/S_{=?}[\mathrm{APP}])), \tag{3}$$

$$\mathrm{ADT} = T \times (1 - R[\text{“weight}_{\mathrm{app}}\text{”}]_{=?}[S] / R[\text{“weight}_{\mathrm{total}}\text{”}]_{=?}[S]), \tag{4}$$

$$\mathrm{WCRT} = \mathrm{filter}(\mathrm{avg}, (R\{\text{“weight}_{\mathrm{fail}}\text{”}\}_{\max=?}[\boldsymbol{F}\ \text{“reboot”}] + \mathrm{reboot\_time}), \mathrm{NMI} \wedge \neg D_1), \tag{5}$$

$$\mathrm{MTTF} = \mathrm{filter}(\mathrm{avg}, (R\{\text{“weight}_{\mathrm{total}}\text{”}\}[\boldsymbol{F}\ \text{“reboot”}])). \tag{6}$$

For details regarding the PRISM specification “filter” “max” “S”, please refer to [5]. Here we use three real-world applications (AProg, BProg, and CProg) to illustrate our method.

Eqs. (1) and (2) are qualitative analyses and Eqs. (3)–(6) are quantitative analyses. Figure 1(c) shows a typical probability of failure on demand (PFD) result comparing the PFMV, RB, and NVP models with non-redundancy. The horizontal axis represents the reliability of a single version ξ (e.g., 95 means $\xi = 95\%$).

• Verifying model correctness. The model checker can find errors such as deadlock, contradiction, and unreachability and can assert the properties specified in the PCTL formulas that relate to the model conformance.

Eq. (1) guarantees that all the states transit to APP (user application) when RST is on. Eq. (2)

ensures that when all the data are bad, the model reboots and returns to the beginning of the model at the time RST.

Although the formulas appear to be relatively simple, they guarantee the essential correctness of the model. We have used these formulas as representative examples and other formulas can be similarly verified.

• Probability of failure on demand (PFD). PFD measures the probability of system failure when requesting a system service. We propose the use of (3) to specify the PFD metric that is the ratio of the long-run probability of operation failure over that of the APP being on.

In Figure 1(c), we compare the results of (3) with respect to NVP, RB, PFMV, and non-redundancy. The baseline (blue) indicates the results when there is no redundancy provided. We can accurately determine the PFD value through the analysis and the results indicate that PFMV achieves dramatic enhancement of the PFD. Moreover, it is also obvious that the evaluation with NVP or RB brings about deviation.

• Average down time (ADT). ADT, as described in (4), is a criterion by which the systems work within an assigned time duration T, e.g., one year (8760 h), and is typically used for evaluating a continuously running system.

After comparing the ADT values obtained via the PFMV, RB, and NVP method, we can deduce that if we use RB for an approximation, the value will be overly assessed from the crossing point $\xi = 82\%$, but insufficiently assessed prior to the crossing point. The NVP approximation, however, always results in over assessment.

• Worse-case response time (WCRT). In real-time systems, computation must guarantee a response within specified time constraints, which are often used to indicate, e.g., a deadline for execution [2]. We employ (5) to calculate the performance of the system in response to a failure in WCRT.

Next, we apply WCRT to AProg, BProg, and CProg with different reboot overheads. We found AProg and CProg to achieve almost the same results, whereas the value of BProg is much larger, indicating that the performance of BProg is worse. These results demonstrate that the WCRT for PFMV is based on the probability of failure and the period time of execution. The CProg results from the NVP and RB models are very close to the horizontal axis.

We obtain a value 112.48 for (5) for AProg. When a train is traveling at velocity of 100 km/h, 112.48 s implies that the train cannot be controlled within a distance of 3124.44 m. Similarly, the approximations using RB or NVP are very dangerous since great deviation leads to the underestimation of reliability.

• Mean time to failure (MTTF). In reliability engineering, the MTTF is an essential index. In our work, we consider the MTTF to be the average time between two reboot events, as given in (6).

Considering that both the reliability of a single version and detection are 0.95, Eq. (6) results for PFMV (AProg) are 0.30, 5.99, and 4499.98 s when the redundancy ranges from 1 to 3, respectively (with similar results for BProg and CProg). As such, we can conclude that the reboot occurrence is tremendously reduced when backup is performed twice. When using RB or NVP to generate estimates, these results also deviate.

Conclusion. In this article, we proposed an evaluation method for a redundancy-based system using probabilistic model checking. This redundant strategy model can be specified accurately using formal language without approximation or simplification, and the verification is supported theoretically. The qualitative and quantitative evaluations can verify numerous temporal computational properties to address the interests of both system designers and users.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61472240, 61672340, 61472238).

References

1 Pham H. Handbook of Reliability Engineering. Berlin: Springer, 2003
2 Birolini A. Reliability Engineering: Theory and Practice. Berlin: Springer, 2010
3 Carzaniga A, Mattevelli A, Pezze M. Measuring software redundancy. In: Proceedings of International Conference on Software Engineering (ICSE15), Florence, 2015. 156–166
4 Baier C, Katoen J. Principles of Model Checking. Cambridge: MIT Press, 2008
5 PRISM website. www.prismmodelchecker.org
6 Wolter K, Avritzer A, Vieira M, et al. Resilience Assessment and Evaluation of Computing Systems. Berlin: Springer, 2012
7 Yiu J. The Definitive Guide to the ARM Cortex-M3. Amsterdam: Elsevier, 2009
8 FUJIMI website. www.letech.jp
9 Kanekawa N, Ibe H, Suga T, et al. Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electromagnetic Disturbances. Berlin: Springer, 2010