Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Performance Evaluation 56 (2004) 249–276

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Performability analysis of guarded-operation duration: a translation approach for reward model solutions

Ann T. Tai [a,*], William H. Sanders [b,1], Leon Alkalai [c,2], Savio N. Chau [c,2], Kam S. Tso [a,3]

[a] *IA Tech, Inc., 10501 Kinnard Avenue, Los Angeles, CA 90024, USA*
[b] *University of Illinois, Urbana, IL 61801, USA*
[c] *Jet Propulsion Laboratory, Pasadena, CA 91109, USA*

Abstract

Performability measures are often defined for analyzing the worth of fault-tolerant systems whose performance is gracefully degradable. Accordingly, performability evaluation is inherently well suited for application of reward model solution techniques. On the other hand, the complexity of performability evaluation for solving engineering problems may prevent us from utilizing those techniques directly, suggesting the need for approaches that would enable us to exploit reward model solution techniques through problem transformation. In this paper, we present a performability modeling effort that analyzes the guarded-operation duration for onboard software upgrading. More specifically, we define a "performability index" Y that quantifies the extent to which the guarded operation with a duration ϕ reduces the expected total performance degradation. In order to solve for Y, we progressively translate its formulation until it becomes an aggregate of constituent measures conducive to efficient reward model solutions. Based on the reward-mapping-enabled intermediate model, we specify reward structures in the composite base model which is built on three stochastic activity network reward models. We describe the model-translation approach and show its feasibility for design-oriented performability modeling.

Keywords: Performability; Reduction of total performance degradation; Duration of guarded operation; Reward model solutions; Model translation; Stochastic activity networks

1. Introduction

In order to protect an evolvable, distributed embedded system for long-life missions against the adverse effects of design faults introduced by an onboard software upgrade, a methodology called

* Corresponding author. Tel.: +1-310-474-3568; fax: +1-310-474-3608.
E-mail addresses: a.t.tai@ieee.org (A.T. Tai), whs@crhc.uiuc.edu (W.H. Sanders), lalkalai@jpl.nasa.gov (L. Alkalai), schau@jpl.nasa.gov (S.N. Chau), k.tso@ieee.org (K.S. Tso).
[1] Tel.: +1-217-333-0345.
[2] Tel.: +1-818-354-3309.
[3] Tel.: +1-310-474-3568.

0166-5316/$ – see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0166-5316(03)00138-X

guarded software upgrading (GSU) has been developed [1–3]. The GSU methodology is supported by a message-driven confidence-driven (MDCD) protocol that enables effective and efficient use of checkpointing and acceptance test techniques for error containment and recovery. More specifically, the MDCD protocol is responsible for ensuring that the system functions properly after a software component is replaced by an updated version during a mission, while allowing the updated component to interact freely with other components in the system. The period during which the system is under the escort of the MDCD protocol is called "guarded operation".

Guarded operation thus permits an upgraded software component to start its service to the mission in a seamless fashion, and, if the escorting process determines that the upgraded component is not sufficiently reliable and thus imposes an unacceptable risk to the mission, ensures that the system will be safely downgraded back by replacing the upgraded software component with an earlier version. It is anticipated that sensible use of this escorting process will minimize the expected total performance degradation, which comprises: (1) the performance penalty due to design-fault-caused failure, and (2) the performance reduction due to the overhead of the safeguard activities. Accordingly, an important design parameter is the duration of the guarded operation ϕ, as the total performance degradation is directly influenced by the length of the escorting process. In turn, this suggests that a performability analysis [4] is pertinent to the engineering decision-making.

Performability modeling often implies the need to consider a broad spectrum of system attributes simultaneously and assess their collective effect on the benefit from the system or the worth of a mission the system intends to accomplish. Accordingly, performability evaluation is inherently well suited for the applications of: (1) reward model solution techniques (see [5–8] for example), (2) methods for hierarchical or hybrid composition (see [9,10] for example), and behavioral decomposition (see [11,12] for example), and (3) tools that implement those modeling techniques (see [13,14] for example). On the other hand, the complexity of performability measures for analyzing engineering problems and the dependencies among the system attributes or subsystems that are subject to a joint consideration may prevent us from exploiting those techniques in a straightforward fashion. Hence, performability analysis with the motivation described in the preceding paragraph presents us with greater challenges than the separate dependability and performance studies for GSU we conducted earlier [2,3].

To address the challenges, we propose a model-translation approach that enables us to exploit reward model solution techniques which we would otherwise be unable to utilize. Rather than attempt to map the performability measure directly to a single reward structure in a monolithic model, we transform the problem of solving a complex performability measure into that of evaluating several constituent reward variables, each of which can be easily mapped to a reward structure and thereby evaluated efficiently using any software tool that supports reward model solutions.

In particular, we first define a "performability index" Y, that quantifies the extent to which the guarded operation with a duration ϕ reduces the expected total performance degradation, relative to the case in which guarded operation is completely absent. For clarity and simplicity of the design-oriented model, we allow Y to be formulated at a high level of abstraction. In order to solve for Y efficiently, we choose not to elaborate its formulation directly or expand the design-oriented model into a monolithic, state-space based model. Instead, we translate the model progressively, through analytic manipulation, into an evaluation-oriented form that is an aggregate of constituent measures conducive to reward model solutions. Based on this intermediate, reward-mapping-enabled model, we take the final step to specify reward structures in the composite base model, which is built on three measure-adaptive stochastic activity network (SAN) [15] reward models.

As with behavioral decomposition methods and hierarchical composition techniques, our model-translation approach permits us to avoid dealing with a model that is too complex to allow direct derivation of a closed-form solution. Whereas the most important relationship between those previously developed techniques and our approach is that successive model translation is intended to enable the application of techniques for reward model solutions, behavioral decomposition, and hierarchical/hybrid composition to performability modeling problems in which: (1) clear boundaries among "subsystems" or system properties could not be perceived from the viewpoint of the original problem formulation, or (2) the mathematical implications (to the performability measure) of system behavior may not become apparent until we elaborate the formulation of the problem to a certain degree. More generally, the process of transforming the problem of solving a complex performability measure into that of evaluating constituent reward variables naturally enables us to utilize those existing, efficient modeling techniques and tools that we would be unable to exploit without model translation, widening the scope of their applicability.

The next section provides an overview of the GSU methodology and a description of guarded operation. Section 3 defines and formulates the performability measure. Section 4 explains the translation process in detail, followed by Section 5, which shows how the reward structures are specified in the SAN models. Section 6 presents an analysis of optimal guarded-operation duration. The paper is concluded in Section 7, which summaries what we have accomplished.

2. Review of guarded software upgrading

The development of the GSU methodology was motivated by the challenge of guarding an embedded system against the adverse effects of design faults introduced by onboard software upgrades [1,3]. The performability study presented in this paper assumes that the underlying embedded system consists of three computing nodes. (This assumption is consistent with the current architecture of the Future Deliveries Testbed at JPL.[4]) Since a software upgrade is normally conducted during a non-critical mission phase when the spacecraft and science functions do not require full computation power, only two processes, corresponding to two different application software components, are supposed to run concurrently and interact with each other. To exploit inherent system resource redundancies, we let the old version, in which we have high confidence due to its sufficiently long onboard execution time, escort the new-version software component through two stages of GSU, namely, *onboard validation* and *guarded operation*, as illustrated in Fig. 1.

Further, we make use of the third processor, which would otherwise be idle during a non-critical mission phase, to accommodate the old version such that the three processes (i.e., the two corresponding to the new and old versions, and the process corresponding to the second application software component) can be executed concurrently. To aid in the description, we introduce the following notation:

P_1^{new}	The process corresponding to the new version of an application software component.
P_1^{old}	The process corresponding to the old version of the application software component.
P_2	The process corresponding to another application software component (which is not undergoing upgrade).

[4] More recently, we have extended the error containment and recovery algorithms so that the methodology can serve a more general class of distributed embedded systems [16].

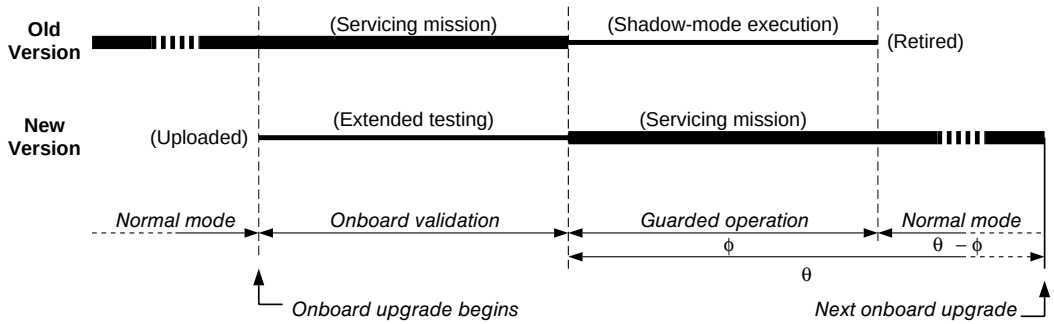


Fig. 1. Onboard guarded software upgrading.

The first stage of GSU (i.e., onboard validation), which can be viewed as extended testing in an actual space environment, starts right after the new version is uploaded to the spacecraft. During this stage, the outgoing messages of the shadow process P_1^{new} are suppressed but selectively logged, while P_1^{new} receives the same incoming messages that the active process P_1^{old} does. Thus, P_1^{new} and P_1^{old} can perform the same computation based on identical input data. By maintaining an onboard error log that can be downloaded to the ground for validation-results monitoring and Bayesian-statistics reliability analyses (as suggested by some prior work in the research literature, see [17] for example), we can make decisions regarding how long onboard validation should continue and whether P_1^{new} can be allowed to enter mission operation. Moreover, onboard extended testing leads to a better estimation of the fault-manifestation rate of the upgraded software. If onboard validation concludes successfully, then P_1^{new} and P_1^{old} switch their roles to enter the guarded operation stage. The time to the next upgrade θ is determined upon the completion of onboard validation, according to: (1) the planned duty of the flight software in the forthcoming mission phases, and (2) the quality of the flight software learned through onboard validation.

During guarded operation, P_1^{new} actually influences the external world and interacts with process P_2 under the escort of the MDCD error containment and recovery protocol, while the messages of P_1^{old} that convey its computation results to P_2 or external systems (e.g., devices and actuators) are suppressed. We call the messages sent by processes to external systems and the messages between processes *external messages* and *internal messages*, respectively.

Because the objective of the MDCD protocol is to mitigate the effect of residual software design faults, we must ensure consistency among different processes' views on verified correctness (validity) of process states and messages. Accordingly, the MDCD algorithms aim to ensure that the error recovery mechanisms can bring the system into a global state that satisfies validity-concerned global state consistency and recoverability. The key assumption used in the derivation of the MDCD algorithms is that an erroneous state of a process is likely to affect the correctness of its outgoing messages, while an erroneous message received by an application software component will result in process state contamination [2]. Accordingly, the necessary and sufficient condition for a process to establish a checkpoint is that the process receives a message that will make the process's otherwise non-contaminated state become potentially contaminated. In order to keep performance overhead low, the correctness validation mechanism, *acceptance test* (AT), is only used to validate external messages from the active processes that are potentially contaminated. By a "potentially contaminated process state", we mean: (1) the process state

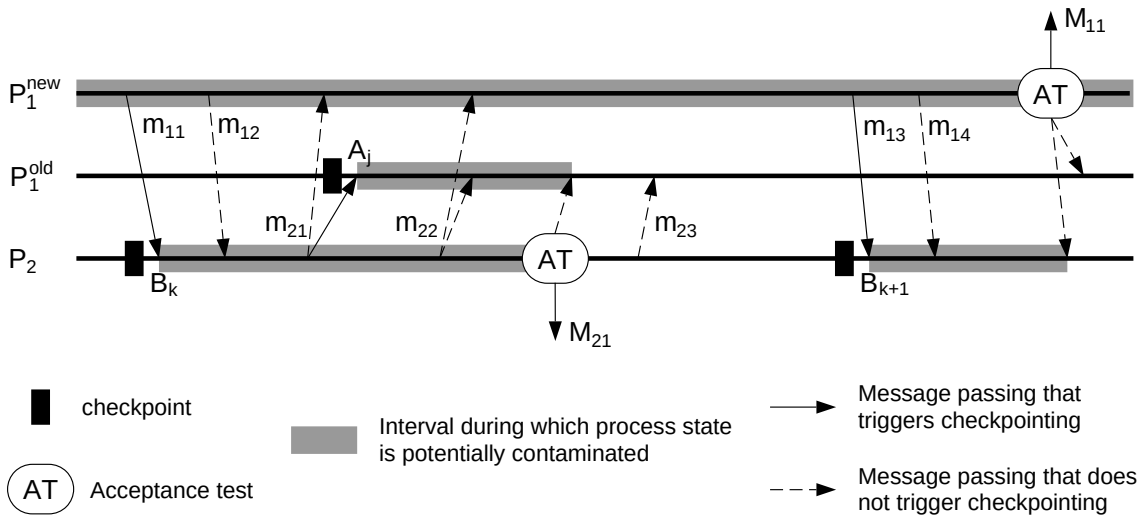


Fig. 2. MDCD protocol for guarded operation.

of P_1^{new} which is created from a low-confidence software component, or (2) a process state that reflects the receipt of a not-yet-validated message that is sent by a process when its process state is potentially contaminated.

Fig. 2 illustrates the behavior of the MDCD protocol. The horizontal lines in the figure represent the software executions along the time horizon. Each of the shaded regions represents an execution interval during which the state of the corresponding process is potentially contaminated. Symbols m_{ij} and M_{ik} denote, respectively, the jth internal message and kth external message sent by process P_i.

Upon the detection of an erroneous external message, P_1^{old} will take over P_1^{new}'s active role and prepare to resume normal computation with P_2. By locally checking its knowledge about whether its process state is contaminated, a process will decide to roll back or roll forward, respectively. After a rollback or roll-forward action, P_1^{old} will "re-send" the messages in its message log or further suppress messages it intends to send, based on the knowledge about the validity of P_1^{new}'s messages. After error recovery (which marks an unsuccessful but safe onboard upgrade), the system goes back to the normal mode (under which safeguard functions, namely, checkpointing and AT, are no longer performed) until the next scheduled upgrade. An undetected, erroneous external message[5] will result in system failure, meaning that the system will become unable to continue proper mission operation. On the other hand, as the MDCD algorithms allow very simple error recovery [3], we anticipate that the system will recover from an error successfully as long as the detection is successful.

If no error occurs during ϕ, then guarded operation concludes and the system goes back to the normal mode (see Fig. 1). Note that while the time to the next scheduled onboard upgrade θ is chosen via a software engineering decision, the duration of guarded operation ϕ is a design parameter that influences system performance and dependability. The central purpose of this paper is to study how to evaluate a performability measure for determining an optimal ϕ. In the section that follows, we define and formulate the performability measure.

[5] For simplicity, in the remainder of the paper, we use the term "error" to refer to an erroneous external message.

3. Performability measure

3.1. Definition

We define a performability measure that will help us choose the appropriate duration of guarded operation ϕ. More specifically, ϕ will be determined based on the value of the performability measure that quantifies the expected total performance degradation reduction resulting from guarded operation.

As mentioned in Section 1, we consider two types of performance degradation, namely

1. the performance degradation due to design-fault-caused failure, and
2. the performance degradation caused by the performance overhead of checkpoint establishment and AT-based validation.

Clearly, a greater value of ϕ implies: (1) a decrease in the expected performance degradation due to potential system failure caused by residual design faults in the upgraded software component, and (2) an increase in the expected performance degradation due to the overhead of checkpointing and AT. We let "mission worth" be quantified by the system time that is devoted to performing application tasks rather than safeguard activities. If we let W_ϕ denote the amount of mission worth that is accrued through θ when the duration of guarded operation (G-OP) is ϕ, then W_0 refers to the total mission worth accrued through θ for the boundary case in which the G-OP mode is completely absent (having a zero duration). On the other extreme, if the system is perfectly reliable, then it would not require guarded operation and would thus be free of either type of performance degradation described above. We view this extreme case as the "ideal case" and let its total mission worth (accrued through θ) be denoted by W_I.

It is worthwhile noting that the difference between the expected values of W_I and W_ϕ can be regarded as the expected mission worth reduction, or the expected total performance degradation (from the ideal case) that the system experiences through θ when the G-OP duration is ϕ. Similarly, the difference between the expected values of W_I and W_0 represents the expected total performance degradation the system experiences through θ when the G-OP mode is absent throughout θ. It follows that if $E[W_I] - E[W_\phi] < E[W_I] - E[W_{\phi'}]$, then ϕ can be considered a better choice than ϕ'. Accordingly, we let the performability measure take the form of a *performability index* Y, that quantifies the extent to which a G-OP duration ϕ reduces the expected total performance degradation, relative to the case in which the G-OP mode is completely absent. More succinctly, Y is the ratio of the difference between $E[W_I]$ and $E[W_0]$ to that between $E[W_I]$ and $E[W_\phi]$:

$$Y = \frac{E[W_I] - E[W_0]}{E[W_I] - E[W_\phi]}. \tag{1}$$

Based on the above discussion, we can anticipate a performability benefit from a guarded operation that is characterized by a (non-zero) duration ϕ when $E[W_I] - E[W_\phi]$ is less than $E[W_I] - E[W_0]$. More precisely, $Y > 1$ implies that the application of guarded operation will yield a performability benefit with respect to the reduction of the expected total performance degradation. On the other hand, $Y \leq 1$ suggests that guarded operation will not be effective for total performance degradation reduction. We formulate $E[W_I]$, $E[W_0]$, and $E[W_\phi]$ in the next section.

3.2. Formulation

As explained above, we choose to quantify "mission worth" in terms of the system time devoted to performing application tasks (rather than safeguard activities) that is accrued through mission period θ. Further, the system behavior described in Section 2 suggests that an error that propagates to an external system will nullify the worth of that mission period. Since neither the ideal case nor the case in which the G-OP mode is completely absent involves safeguard activities, W_{I} and W_0 can be formulated in a straightforward fashion:

$$W_{\mathrm{I}} = 2\theta, \tag{2}$$

$$W_0 = \begin{cases} 2\theta & \text{if no error occurs throughout } \theta, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that the coefficient 2 in the above equations reflects the fact that in the avionics system we consider, only two application processes actively service the mission during θ. (For the cases to which W_{I} and W_0 correspond, the two processes will always be $\mathrm{P}_1^{\mathrm{new}}$ and P_2.)

To help formulate an expression for W_ϕ, we group the possible behaviors (i.e., sample paths) that the system may exhibit into several categories. In particular, since we do not make the assumption that $\mathrm{P}_1^{\mathrm{old}}$ and P_2 are perfectly reliable and AT has a full coverage, we must consider situations where the system fails during guarded operation, or fails after error recovery. This leads us to define three classes of sample paths: (i) those (called S1 below) in which no error occurs, and the system thus goes through the upgrade process successfully, (ii) those (called S2 below) in which an error occurs during ϕ, and the system successfully recovers, and (iii) those involving the occurrence of an error from which the system cannot recover. (Since no mission worth is accumulated for sample paths of the third category, they are not considered in the expressions of mission worth.) More specifically, we define sets of sample paths S1 and S2 as follows:

(S1) No error occurs by the end of ϕ, so the system enters the normal mode with $\mathrm{P}_1^{\mathrm{new}}$ and P_2 in mission operation after ϕ; the upgraded system subsequently goes through the period $(\theta - \phi)$ successfully.

(S2) An error occurs and is detected by $\mathrm{P}_1^{\mathrm{new}}$ or P_2 at τ, $0 < \tau \leq \phi$, so that error recovery brings the system into the normal mode with $\mathrm{P}_1^{\mathrm{old}}$ and P_2 in mission operation after τ; the recovered system subsequently goes through $(\theta - \tau)$ successfully.

We let $\rho_{t,1}$ and $\rho_{t,2}$ denote the fractions of time during which $\mathrm{P}_1^{\mathrm{new}}$ and P_2, respectively, are making forward progress (rather than performing safeguard functions), given that the system is under the G-OP mode until t ($t \leq \phi$). W_ϕ can then be defined as follows:

$$W_\phi = \begin{cases} (\rho_{\phi,1} + \rho_{\phi,2})\phi + 2(\theta - \phi) & \text{if system experiences a sample path in S1,} \\ \gamma((\rho_{\tau,1} + \rho_{\tau,2})\tau + 2(\theta - \tau)) & \text{if system experiences a sample path in S2,} \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where the coefficient γ ($0 < \gamma < 1$) is the *discount factor* that takes into account the additional mission worth reduction for an unsuccessful but safe onboard software upgrade, relative to the case in which the upgrade succeeds. We can define γ according to the implication of S2 for the system in question. For clarity of illustration, we will postpone our description of how we define γ until Section 6, in which we present the evaluation experiments and results.

3.3. High-level elaboration of expressions

In order to solve for Y, we first elaborate its formulation at a high level of abstraction. It is clear that $E[W_I] = 2\theta$. Thus we proceed to elaborate $E[W_0]$, the expected value of mission worth when G-OP duration is zero. It is worth noting that when $\phi = 0$, the sample-path set S2 becomes degenerate while S1 is reduced to a "boundary-case version" that enumerates the system behaviors belonging to the category "no error occurs throughout θ when the G-OP mode is completely absent during θ". Then, from Eq. (3), which implies that mission worth will be 2θ if the system takes a sample path in this boundary-case version of S1, it follows that

$$E[W_0] = 2\theta P(\text{S1, when } \phi = 0). \tag{5}$$

Based on Eq. (4) and the theorem of total expectation, we derive the following expression for $E[W_\phi]$:

$$E[W_\phi] = Y_\phi^{S1} + Y_\phi^{S2}, \tag{6}$$

where

$$\begin{aligned} Y_\phi^{S1} &= E[(\rho_{\phi,1} + \rho_{\phi,2})\phi + 2(\theta - \phi)]P(\text{S1, when } \phi > 0) \\ &= ((\rho_{\phi,1} + \rho_{\phi,2})\phi + 2(\theta - \phi))\,P(\text{S1, when } \phi > 0). \end{aligned} \tag{7}$$

We notice that the application-purpose message-passing events that trigger checkpointing and AT (which dominate the performance overhead) are significantly more frequent than the fault-manifestation events. Moreover, the mean time between message-passing events is only seconds in length, whereas a reasonable value of ϕ will be in the range of hundreds or thousands of hours. Hence, we can assume that the system reaches a steady state with respect to the performance-overhead related events before an error occurs or the G-OP duration ends. Thus, $\rho_{t,1}$ and $\rho_{t,2}$ can be regarded as steady-state measures ρ_1 and ρ_2, respectively. Consequently, Eq. (7) becomes

$$Y_\phi^{S1} = ((\rho_1 + \rho_2)\phi + 2(\theta - \phi))\,P(\text{S1, when } \phi > 0). \tag{8}$$

While the fact that all the sample paths in S1 (when ϕ takes a particular value in $[0, \theta]$) will yield the same mission worth leads to a simple expression for Y_ϕ^{S1}, the mission worth associated with the sample paths in S2 are a function of τ, which is a random variable that can assume a continuum of values. This precludes the possibility of deriving an expression for Y_ϕ^{S2} that is in a form similar to that of Eq. (8). Accordingly, we postulate that there exists a closed-form solution for Y_ϕ^{S2} and elaborate it by defining two probability density functions (pdf). More precisely, we let h be the pdf of τ, and let f denote the pdf of the time to system failure that occurs after error recovery. As f is conditioned on the event that the system has recovered from an error that is detected successfully at τ, $0 < \tau \leq \phi$, Y_ϕ^{S2} can be formulated as follows:

$$Y_\phi^{S2} = \gamma \int_0^\phi ((\rho_1 + \rho_2)\tau + 2(\theta - \tau))h(\tau)\left(1 - \int_\tau^\theta f(x)\,\mathrm{d}x\right)\mathrm{d}\tau, \tag{9}$$

where the term $(1 - \int_\tau^\theta f(x)\,\mathrm{d}x)$ is the probability that system failure does not occur in the interval from error detection to the next onboard upgrade (during this interval P_1^{old} and P_2 are in mission operation), given that an error is detected and recovered at τ, $0 < \tau \leq \phi$.

We now arrive at a stage in which Y is formulated in not-yet-elaborated pdf. The decision of how to proceed from there is important. We choose not to elaborate those pdf, namely h and f, because they

would have complex expressions and would thus make the solution of Eq. (9) very difficult. Rather, in order to take advantage of reward model solution techniques and tools that support them, we prefer to let those terms that remain at a high level of abstraction guide us in reward structure derivation. Nonetheless, neither Eq. (8) nor Eq. (9) has been stated in a form that is ready to be mapped to reward structures in a state-space based model. Accordingly, we seek to exploit reward model solution techniques through model translation.

4. Translation for reward model solutions

With the motivation described at the end of the previous section, we develop an approach that translates the design-oriented model successively until it reaches a stage at which the final solution of Y becomes a simple function of "constituent measures", each of which can be directly mapped to a reward structure. Fig. 3 illustrates the process of successive model translation. As shown by the diagram, translation proceeds along two branches: one for solving $E[W_0]$ and one for solving $E[W_\phi]$ (which is expressed as the sum of Y_ϕ^{S1} and Y_ϕ^{S2}). As indicated by Eqs. (5), (8) and (9), solution derivations for $E[W_0]$ and Y_ϕ^{S1} will deal solely with probabilistic measures concerning the sample paths in S1, whereas the terms that are involved in Y_ϕ^{S2} characterize the system behavior constituting S2. We describe the translation process for solving $E[W_0]$ and Y_ϕ^{S1} in Section 4.1, and devote Section 4.2 to the explanation of how we translate Y_ϕ^{S2} step by step.

4.1. Translation based on sample-path analysis

Note that the high-level expressions that we have derived for $E[W_0]$ and $E[W_\phi]$ in Section 3.3 indeed postulate a stochastic process with a sample-path space that covers S1 and S2. If we view the starting point of G-OP as time zero (see Fig. 1), this stochastic process can be expressed as $\mathcal{X} = \{X_t | t \in [0, \theta]\}$. Further, if we let $\mathcal{A}_1$ denote the set of states of $\mathcal{X}$ in which no error has occurred in the system, then according to the definition of S1:

$$P(\text{S1, when } \phi = 0) = P(X_\theta \in \mathcal{A}_1, \text{ when } \phi = 0), \tag{10}$$

$$P(\text{S1, when } \phi > 0) = P(X_\theta \in \mathcal{A}_1, \text{ when } \phi > 0). \tag{11}$$

It is possible to specify a monolithic model to represent the stochastic process $\mathcal{X}$ for solving $P(X_\theta \in \mathcal{A}_1, \text{ when } \phi = 0)$ and $P(X_\theta \in \mathcal{A}_1, \text{ when } \phi > 0)$, if we choose to use a model type that is highly expressive, such as stochastic Petri nets or SANs. However, the complexity of the model would make it impossible for us to achieve solution efficiency, even if the model was comprehensive enough to support the measures. Specifically, in addition to the problem of state-space size, the fact that $\mathcal{X}$ is characterized by a pre-designated G-OP duration ϕ implies the inclusion of a deterministic state transition which prevents $\mathcal{X}$ from being Markovian.

On the other hand, since ϕ is a crucial design parameter whose value determines the transition point of system operation and is supposed to have a significant impact on performability, we can translate the model by letting ϕ be the cutoff point in breaking down each of the sample paths. More specifically, if we let $\overline{U}_{[0,\phi]}$ denote the sample-path set that enumerates the system's behavior within the interval defined

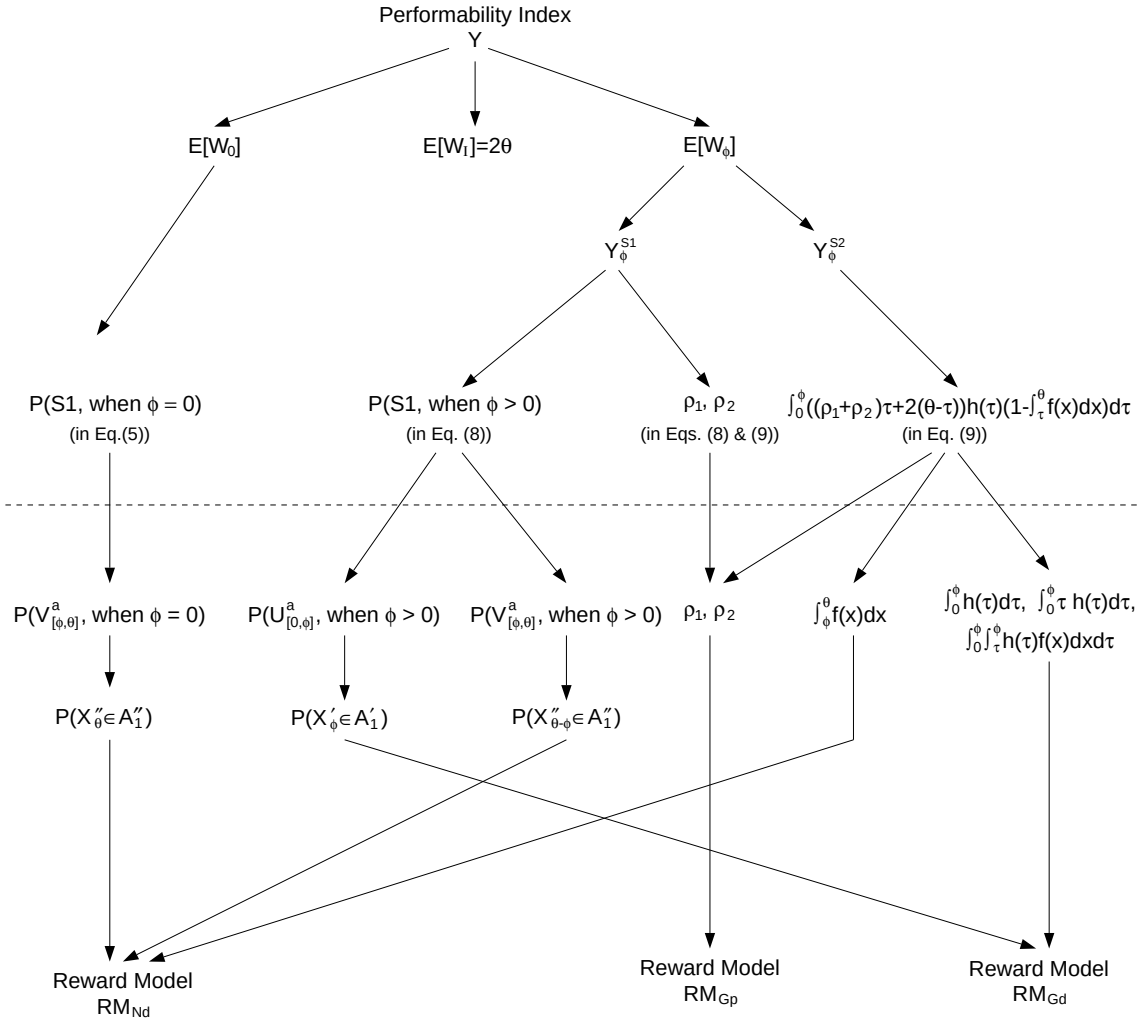


Fig. 3. Successive model translation.

by the starting point of G-OP and the point corresponding to the pre-designated value of ϕ, then $\overline{U}_{[0,\phi]}$ is composed of three subsets:

$$\overline{U}_{[0,\phi]} = \{U^{\mathrm{a}}_{[0,\phi]}, U^{\mathrm{b}}_{[0,\phi]}, U^{\mathrm{c}}_{[0,\phi]}\},$$

where the sample paths in the subsets cover the following categories of system behavior:

$U^{\mathrm{a}}_{[0,\phi]}$ The system goes through the entire length of $[0, \phi]$ successfully under the G-OP mode with no error occurrence.

$U^{\mathrm{b}}_{[0,\phi]}$ An error occurs and is detected at τ ($\tau \leq \phi$); the recovery system goes through $(\tau, \phi]$ successfully under the normal mode.

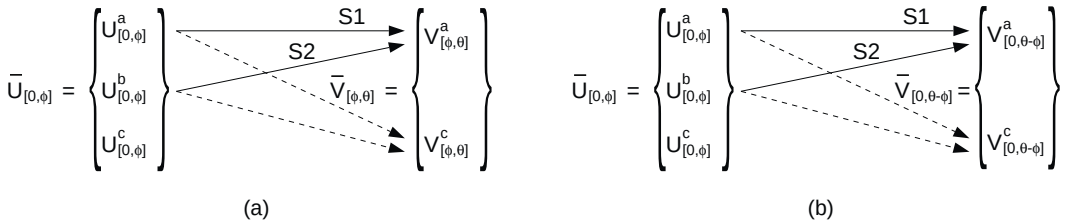


Fig. 4. Composite sample paths.

$U^{c}_{[0,\phi]}$ An error occurs in $(0, \phi]$ but goes undetected, resulting in an immediate system failure, or an error occurs and is detected at τ ($\tau \leq \phi$), but the recovered system fails (under the normal mode) in the interval $(\tau, \phi]$ due to another error.

Similarly, we let $\overline{V}_{[\phi,\theta]}$ denote the sample-path set that enumerates the system's behavior within the interval defined by the point corresponding to the pre-designated value of ϕ and θ (the point for the next upgrade). Thus $\overline{V}_{[\phi,\theta]}$ consists of two subsets:

$$\overline{V}_{[\phi,\theta]} = \{V^{a}_{[\phi,\theta]}, V^{c}_{[\phi,\theta]}\},$$

where the sample paths in the subsets cover the following types of system behavior under the normal mode (given that the system does not fail by ϕ):

$V^{a}_{[\phi,\theta]}$ The system goes through the period $[\phi, \theta]$ successfully.
$V^{c}_{[\phi,\theta]}$ The system fails during $[\phi, \theta]$ due to an error that occurs during that period.

Then, the sample-path space of $\mathcal{X}$, call it S, is a proper subset of $\overline{U}_{[0,\phi]} \times \overline{V}_{[\phi,\theta]}$. More succinctly, $S \subset \overline{U}_{[0,\phi]} \times \overline{V}_{[\phi,\theta]}$, as illustrated in Fig. 4(a). Note that all the composite sample paths indicated by the dashed lines with arrows and the sample paths in $U^{c}_{[0,\phi]}$ imply the absorbing states of $\mathcal{X}$. (Therefore, no mission worth will be accumulated through those paths.)

The above analysis suggests that we can decompose $\mathcal{X}$ with respect to the pre-designated value of ϕ. More precisely, $\mathcal{X}$ can be partitioned into two simpler stochastic processes, $\mathcal{X}' = \{X'_t | t \in [0, \phi]\}$ and $\mathcal{X}'' = \{X''_t | t \in [\phi, \theta]\}$. The former represents system behavior during the interval with a pre-designated length ϕ ($\phi > 0$, but the system may switch from the G-OP mode to normal mode at τ, $0 < \tau \leq \phi$, due to error recovery), while the latter represents system behavior from ϕ to the point at which the next onboard upgrade is supposed to begin, provided that the system does not fail during $[0, \phi]$.

Accordingly, $\overline{U}_{[0,\phi]}$, which exhaustively enumerates all the possible system behaviors during the interval $[0, \phi]$, becomes the sample-path space of $\mathcal{X}'$; likewise, $\overline{V}_{[\phi,\theta]}$ is the sample-path space of $\mathcal{X}''$.

Note that for the boundary case in which $\phi = 0$, $\mathcal{X}'$ becomes degenerate while $\mathcal{X}''$ becomes the model that represents the system behavior when G-OP is absent in the entire duration of $[0, \theta]$ (i.e., the case upon which W_0 is defined). Without losing generality, when $\phi = 0$, $V^{a}_{[\phi,\theta]}$ and $V^{c}_{[\phi,\theta]}$ can be interpreted, respectively, as the sets of sample paths representing the scenarios in which: (1) a completely unprotected system operates properly, and (2) an error causes the unprotected system to fail.

As revealed by the reliability study we conducted in [2], when processes' message-sending rates are reasonably high, the likelihood that dormant error conditions will exist in a process state upon a successful

completion of G-OP or after error recovery is so low that the effect on system behavior is practically negligible. This result implies that each process involved in mission operation at ϕ can be considered as "clean" as it was at time zero. Thus $V^{a}_{[\phi,\theta]}$ and $V^{c}_{[\phi,\theta]}$ are approximately equivalent to $V^{a}_{[0,\theta-\phi]}$ and $V^{c}_{[0,\theta-\phi]}$, respectively. In turn, this means that $\mathcal{X}'' = \{X''_t | t \in [\phi, \theta]\}$ can be treated as $\mathcal{X}'' = \{X''_t | t \in [0, \theta - \phi]\}$. Further, as $\phi \in [0, \theta]$, the maximum value of $(\theta - \phi)$ will be θ. Consequently, this stochastic process can be expressed as $\mathcal{X}'' = \{X''_t | t \in [0, \theta]\}$.

As a result of the decomposition, S1 and S2 can be expressed as the Cartesian products of a subset from $\overline{U}_{[0,\phi]}$ and a subset from $\overline{V}_{[0,\theta-\phi]}$. Specifically, as indicated in Fig. 4(b), when $\phi > 0$, each sample path in S1 can be regarded as a path from $U^{a}_{[0,\phi]}$ concatenated with a path from $V^{a}_{[0,\theta-\phi]}$; but when $\phi = 0$, S1 simply equals $V^{a}_{[0,\theta]}$. More succinctly:

$$\mathrm{S1} = \begin{cases} U^{a}_{[0,\phi]} \times V^{a}_{[0,\theta-\phi]} & \text{if } \phi > 0, \\ V^{a}_{[0,\theta]} & \text{if } \phi = 0. \end{cases} \tag{12}$$

Likewise

$$\mathrm{S2} = U^{b}_{[0,\phi]} \times V^{a}_{[0,\theta-\phi]} \quad \text{if } \phi > 0. \tag{13}$$

Clearly, S2 becomes degenerate if $\phi = 0$.

The translation-induced model decomposition enables us to solve for $P(\mathrm{S1}, \text{when } \phi = 0)$ and $P(\mathrm{S1}, \text{when } \phi > 0)$ in an efficient way. Specifically, if we let $\mathcal{A}'_1$ and $\mathcal{A}''_1$ denote, respectively, the sets of states of $\mathcal{X}'$ and $\mathcal{X}''$ in which no error has occurred in the system, then Eqs. (10) and (11) can be translated into a combined expression:

$$P(\mathrm{S1}) = \begin{cases} P(X'_\phi \in \mathcal{A}'_1) P(X''_{\theta-\phi} \in \mathcal{A}''_1) & \text{if } \phi > 0, \\ P(X''_\theta \in \mathcal{A}''_1) & \text{if } \phi = 0. \end{cases} \tag{14}$$

Consequently, each of those transient, instant-of-time measures, namely, $P(X'_\phi \in \mathcal{A}'_1)$, $P(X''_{\theta-\phi} \in \mathcal{A}''_1)$, and $P(X''_\theta \in \mathcal{A}''_1)$, can be solved by defining a reward structure in one of the decomposed models.

As explained in Section 3.3, ρ_1 and ρ_2 can be treated as steady-state instant-of-time measures. This suggests that we can take one step further in model decomposition by partitioning $\mathcal{X}'$ into two submodels, namely, one (with absorbing states) that represents the dependability attributes of $\mathcal{X}'$ and another (having no absorbing states) that represents the performance aspects of $\mathcal{X}'$. By defining reward structures in the latter and computing the expected reward at steady state, we will be able to evaluate ρ_1 and ρ_2.

As indicated by Fig. 3, based on the reward model solutions of constituent measures $P(X'_\phi \in \mathcal{A}'_1)$, $P(X''_{\theta-\phi} \in \mathcal{A}''_1)$, $P(X''_\theta \in \mathcal{A}''_1)$, ρ_1, and ρ_2, we will be able to evaluate $E[W_0]$ and Y^{S1}_ϕ in a straightforward fashion. On the other hand, the model decomposition explained above is helpful but not enough for translating Y^{S2}_ϕ into a form that is conducive to a reward model solution, because as mentioned in Section 3.3, the mission worth associated with each sample path in S2 is a function of τ, which is a random variable that can assume a continuum of values. Hence, we proceed to manipulate the double integral in Eq. (9) analytically, as described in detail in the next section.

4.2. Translation of Y^{S2}_ϕ

The translation of Y^{S2}_ϕ aims to ensure that (1) its final form involves only simple, easily interpretable constituent measures, and (2) none of the resulting constituent measures concerns system behavior going

across the boundary point ϕ. We begin with expanding the right hand side of Eq. (9) as follows:

$$Y_{\phi}^{S2} = \gamma \left(\int_0^{\phi} (2\theta - (2 - (\rho_1 + \rho_2))\tau) h(\tau)\, d\tau - \int_0^{\phi} (2\theta - (2 - (\rho_1 + \rho_2))\tau) h(\tau) \int_{\tau}^{\theta} f(x)\, dx\, d\tau \right). \tag{15}$$

If we then rearrange the first term in the parentheses of Eq. (15), we have

$$\int_0^{\phi} (2\theta - (2 - (\rho_1 + \rho_2))\,\tau) h(\tau)\, d\tau = 2\theta \int_0^{\phi} h(\tau)\, d\tau - (2 - (\rho_1 + \rho_2)) \int_0^{\phi} \tau h(\tau)\, d\tau. \tag{16}$$

Note that τ has a mixture distribution. This is because $h(\tau)$ equals zero for $\tau > \phi$ and thus $\lim_{\tau\to\infty} H(\tau) < 1$. Clearly, $\int_0^{\phi} h(\tau)\, d\tau$ is the probability that an error occurs and is detected by ϕ when the G-OP duration is ϕ. However, as mentioned earlier, the complexity of the system behavior makes it very difficult to derive h and compute the integrals without an excessive amount of approximation. Therefore, we choose not to elaborate h but let it guide us to reach reward model solutions. Specifically, per the interpretation of $\int_0^{\phi} h(\tau)\, d\tau$, if we let $\mathcal{A}'_3$ denote the set of states (of $\mathcal{X}'$) in which an error has occurred and been successfully detected, $\int_0^{\phi} h(\tau)\, d\tau$ can then be evaluated as the expected instant-of-time reward:

$$\int_0^{\phi} h(\tau)\, d\tau = P(X'_{\phi} \in \mathcal{A}'_3). \tag{17}$$

In other words, with a state-space based model $\mathcal{X}'$, we can solve $\int_0^{\phi} h(\tau)\, d\tau$ by assigning a reward rate of 1 to all states in $\mathcal{A}'_3$ and a reward rate of zero to all other states, and computing the expected reward at ϕ.

Recall that the system behavior implies that: (1) a state in which the system encounters an undetected error is absorbing, and (2) a successful error detection will result in error recovery that brings the system back to the normal mode, under which checkpointing and AT (the error detection mechanism) will no longer be performed. Together with the fact that $h(\tau) = 0$ for $\tau \in (\phi, \infty)$, these in turn suggest that mean time to error detection (measured from the starting point of G-OP) $\int_0^{\phi} \tau h(\tau)\, d\tau$ is a meaningful measure, and can have a reward model solution. Accordingly, if we let $\mathcal{A}'_2$ denote the set of states in which no error has been detected, and $\mathcal{A}'_4$ denote the set of (absorbing) states in which an error has occurred and caused a system failure due to unsuccessful error detection (thus $\mathcal{A}'_4$ is a proper subset of $\mathcal{A}'_2$), we have

$$\int_0^{\phi} \tau h(\tau)\, d\tau = \int_0^{\phi} (P(X'_t \in \mathcal{A}'_2) - P(X'_t \in \mathcal{A}'_4))\, dt, \tag{18}$$

which implies that to evaluate $\int_0^{\phi} \tau h(\tau)\, d\tau$, we can assign a reward rate of 1 to all states (of $\mathcal{X}'$) in $\mathcal{A}'_2$, a reward rate of -1 to all states in $\mathcal{A}'_4$, and a reward rate of zero to all other states, and then compute the expected reward accumulated through ϕ. Thus, the integrals in Eq. (16) (and thus the minuend in Eq. (15)) can be solved.

We manipulate the subtrahend in Eq. (15) in a similar fashion and begin with rearranging the terms:

$$\begin{aligned} &\int_0^{\phi} (2\theta - (2 - (\rho_1 + \rho_2))\tau) h(\tau) \int_{\tau}^{\theta} f(x)\, dx\, d\tau \\ &\quad = 2\theta \int_0^{\phi} \int_{\tau}^{\theta} h(\tau) f(x)\, dx\, d\tau - (2 - (\rho_1 + \rho_2)) \int_0^{\phi} \int_{\tau}^{\theta} \tau h(\tau) f(x)\, dx\, d\tau \approx 2\theta \int_0^{\phi} \int_{\tau}^{\theta} h(\tau) f(x)\, dx\, d\tau. \end{aligned} \tag{19}$$

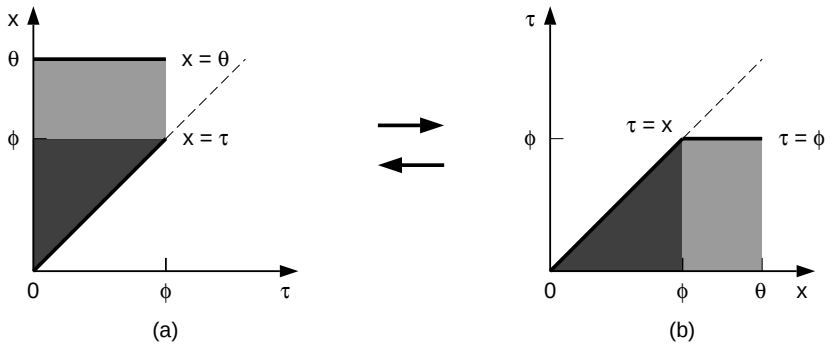


Fig. 5. Translating the area of integration.

We neglect the subtrahend in the above equation because the values of both ρ_1 and ρ_2 will normally (and preferably) be close to unity, while the value of θ in the minuend in the same equation is supposed to be in the range of 1000–10,000 h for the considered applications.

Note also that the expression $\int_0^{\phi}\int_{\tau}^{\theta} h(\tau)\,f(x)\,\mathrm{d}x\,\mathrm{d}\tau$ is the probability that an error occurs and is detected when the system is under the G-OP mode but the system fails due to another error that occurs between the successful recovery and the next upgrade. However, the area of integration (with respect to the time x at which a recovered system fails) goes across the boundary between the pre-designated G-OP interval $[0, \phi]$ and the interval $[\phi, \theta]$ during which the system (that completes G-OP safely) continues mission operation under the normal mode. That prevents us from obtaining a reward model solution based on the decomposed models $\mathcal{X}'$ and $\mathcal{X}''$. However, by closely inspecting the area of the integration, as shown in Fig. 5(a), we realize that we can change the coordinates of the integrals such that the orientation of the integration area will be converted accordingly, as shown in Fig. 5(b). In turn, the converted integration area suggests that the result of Eq. (19) can be broken down into two terms:

$$2\theta\int_0^{\phi}\int_{\tau}^{\theta} h(\tau)\,f(x)\,\mathrm{d}x\,\mathrm{d}\tau = 2\theta\int_0^{\phi}\int_0^{x} f(x)h(\tau)\,\mathrm{d}\tau\,\mathrm{d}x + 2\theta\int_{\phi}^{\theta} f(x)\int_0^{\phi} h(\tau)\,\mathrm{d}\tau\,\mathrm{d}x. \tag{20}$$

The second summand in Eq. (20) can thus be expressed as the product of two probabilities, namely, $\int_0^{\phi} h(\tau)\,\mathrm{d}\tau$ and $\int_{\phi}^{\theta} f(x)\,\mathrm{d}x$ (see Eq. (21)). While we have already provided a reward model solution for the former (see Eq. (17)), we recognize that the latter is the probability that the recovered system will fail due to the occurrence of another error at a time instant in $[\phi, \theta]$. As explained in Section 4.1, we can obtain a good approximation for $\int_{\phi}^{\theta} f(x)\,\mathrm{d}x$ by defining a reward structure in $\mathcal{X}''$ and computing the expected instant-of-time reward at $(\theta - \phi)$.

On the other hand, the first summand in Eq. (20) is not yet in a form that can be interpreted in a straightforward way. We therefore "change back" the coordinates for this individual constituent measure so that its area of integration is translated from the darker region (with a triangular shape) in Fig. 5(b) back to the darker region in Fig. 5(a). In turn, the subtrahend in Eq. (15) finally becomes

$$\begin{aligned}&\int_0^{\phi}(2\theta - (2 - (\rho_1 + \rho_2))\tau)h(\tau)\int_{\tau}^{\theta} f(x)\,\mathrm{d}x\,\mathrm{d}\tau\\ &\quad\approx 2\theta\int_0^{\phi}\int_{\tau}^{\phi} h(\tau)\,f(x)\,\mathrm{d}x\,\mathrm{d}\tau + 2\theta\left(\int_0^{\phi} h(\tau)\,\mathrm{d}\tau\right)\left(\int_{\phi}^{\theta} f(x)\,\mathrm{d}x\right).\end{aligned} \tag{21}$$

Although the first summand in the above equation also involves a double integral, its area of integration is bounded by ϕ. In addition, this double integral can be given a straightforward interpretation, i.e., the probability that an error is detected when the system is under the G-OP mode and the recovered system fails by ϕ (under the normal mode) due to the occurrence of another error. A reward structure can then be defined accordingly in $\mathcal{X}'$. To this end, we can evaluate each of the constituent measures for Y_ϕ^{S2} by mapping it to a reward structure in $\mathcal{X}'$ or $\mathcal{X}''$. More succinctly, if we plug the results of Eqs. (16) and (21) into Eq. (15), Y_ϕ^{S2} becomes ready to be solved using reward model solution techniques.

In summary, the translation process described thus far converts the terms at the design-oriented level into the following solvable constituent reward variables (see also Fig. 3):

- $P(X''_\theta \in \mathcal{A}''_1)$, $P(X'_\phi \in \mathcal{A}'_1)$, and $P(X''_{\theta-\phi} \in \mathcal{A}''_1)$.
- ρ_1, ρ_2.
- $\int_0^\phi h(\tau)\,\mathrm{d}\tau$, $\int_0^\phi \tau h(\tau)\,\mathrm{d}\tau$, $\int_0^\phi \int_\tau^\phi h(\tau)\,f(x)\,\mathrm{d}x\,\mathrm{d}\tau$, and $\int_\phi^\theta f(x)\,\mathrm{d}x$.

It becomes apparent that we will be able to obtain the final solution of Y if we construct the following three reward models at the base-model level:

$\mathrm{RM_{Gd}}$	A reward model that represents the system behavior during the pre-designated G-OP interval and supports dependability measures (i.e., a submodel of $\mathcal{X}'$ constructed for solving dependability measures).
$\mathrm{RM_{Gp}}$	A reward model that represents the system behavior under the G-OP mode and supports performance-overhead measures (i.e., a submodel of $\mathcal{X}'$ constructed for solving performance-overhead measures).
$\mathrm{RM_{Nd}}$	A reward model that represents the system behavior under the normal mode.

Details about the mapping between the resulting constituent measures and the reward structures in $\mathrm{RM_{Gd}}$, $\mathrm{RM_{Gp}}$, and $\mathrm{RM_{Nd}}$ are provided in the next section.

5. SAN reward model solutions for constituent measures

We use SANs to realize the final step of model translation. This choice is based on the following factors: (1) SANs have high-level language constructs that facilitate marking-dependent model specifications and representation of dependencies among system attributes, and (2) the *UltraSAN* tool provides convenient specification capabilities for defining reward structures [13], and (3) by adopting and making minor modifications to the SAN models we developed for our previous (separate) dependability and performance-cost studies [2,3], we will be able to use them as the reward models $\mathrm{RM_{Gd}}$, $\mathrm{RM_{Gp}}$, and $\mathrm{RM_{Nd}}$. In the following sections, we briefly describe these SAN reward models and show the specifications of the SAN-based reward structures which lead to the final solution of the performability measure.

5.1. SAN reward models

The rich syntax and marking-dependent specification capability of SANs allow us to specify every aspect of the protocol precisely. However, we may encounter a state-space explosion or experience very

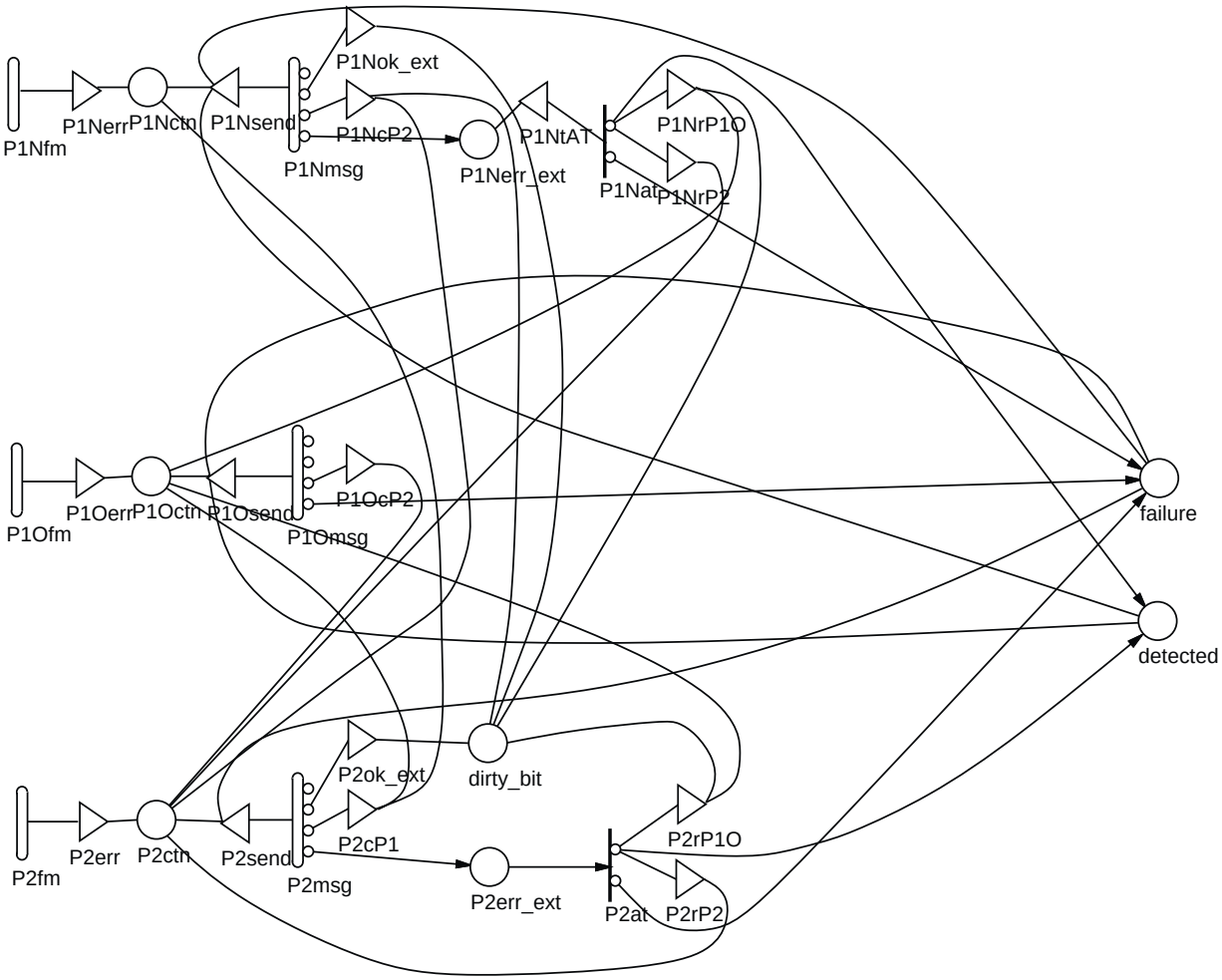


Fig. 6. SAN reward model RM_{Gd}.

slow computation if we attempt to construct a monolithic model, or attempt to make a SAN model a procedural specification of the MDCD protocol. To avoid those problems, we

1. use three separate reward models, each of which is specified for representing the dependability or performance-overhead related aspects of $\mathcal{X}'$ or $\mathcal{X}''$, as explained in Section 4, and
2. minimize explicit representation of the algorithmic details, while ensuring that every aspect of their impact on the particular measure we seek to solve (in a particular reward model) is captured.

The SAN reward model RM_{Gd}, shown in Fig. 6, is a modified version of the model we built for studying the dependability gain from the use of the MDCD protocol [2].

In model construction, we avoid modeling details about checkpoint establishment, deletion, and roll-back error recovery. Rather, by exploiting the relations among the markings of the places that represent

whether a process is actually error-contaminated and the process's knowledge about its state contamination, we are able to characterize the system's failure behavior precisely with respect to whether messages sent by potentially contaminated processes will cause system failure. Consider the output gates `P1Nok_ext` and `P2ok_ext`, which are connected, respectively, to the cases of the timed activities `P1Nmsg` and `P2msg` that represent successful external message sending. The output functions of these two gates will reset the marking of the place `dirty_bit` to zero, implying that the process states of P_2 and P_1^{old} are considered non-contaminated after a message-passing event. These carefully specified cases and output gates play an important role in accurately and concisely representing the system behavior in a non-ideal execution environment. In particular, by resetting `dirty_bit` (through the output function of `P1Nok_ext` or `P2ok_ext`) while leaving the markings of `P1Octn` and `P2ctn` unchanged, we are able to enumerate the following scenarios without introducing separate representations for them:

1. A process that is considered potentially contaminated is actually not contaminated (by own-fault-caused error or error propagation), and thus its external message results in a successful AT.
2. A process that is considered potentially contaminated is actually contaminated, but its error condition is not manifested in the external message, thus after that message passes AT, the process state of P_2 or P_1^{old} that is considered potentially contaminated prior to the AT is wrongly judged non-contaminated.
3. A process that is considered non-contaminated (and may or may not be actually contaminated) sends a correct external message without undergoing AT.

A detailed description of how we extensively exploit marking-dependent specification to represent complex system behavior in a compact model can be found in [2]. In order to represent explicitly whether an error has been detected in the system, the place `detected` is added to the SAN model. Thus, each of the constituent measures that are related to the event of error detection, namely, $\int_0^\phi h(\tau)\,\mathrm{d}\tau$, $\int_0^\phi \tau h(\tau)\,\mathrm{d}\tau$, and $\int_0^\phi \int_\tau^\phi h(\tau) f(x)\,\mathrm{d}x\,\mathrm{d}\tau$, can be easily mapped to a reward structure. Note that since mean time to error occurrence is at least several orders of magnitude greater than the average length of the interval from the start of an AT execution to its completion, we use an instantaneous activity in the SAN reward model RM_{Gd} (which is intended to support constituent measures concerning dependability) to represent an AT-based validation, further reducing model complexity.

In contrast, in the SAN reward model RM_{Gp} (see Fig. 7), we omit those failure-behavior-related aspects, such as fault manifestation, error propagation, and unsuccessful error detection [3]. Instead, we focus on representing the conditions that would require a process to take actions that do not belong to the category of "forward progress" (e.g., the action to establish a checkpoint or perform an AT).

Since its purpose is to solve ρ_1 and ρ_2, the SAN reward model RM_{Gp} includes the timed and instantaneous activities that represent the error containment actions of the protocol that are driven by the message-passing events and the dynamically adjusted confidence in processes, as shown in Fig. 7. In particular, the places `P1oDB` and `P2DB` represent the dirty bits of P_1^{old} and P_2, respectively. Each of those places may have a marking of 0 or 1, which can be interpreted as the confidence in the corresponding process. The timed and instantaneous activities together precisely represent how the MDCD checkpointing rule and AT-based validation policy are executed. For example, when activated, the timed activity `P2_CKPT` indicates that P_2 is engaged in a checkpoint establishment, while the instantaneous activity `P2SkipCKPT` indicates that P_2 is exempted from checkpointing for a particular message-receiving event, according to the MDCD checkpointing rule.

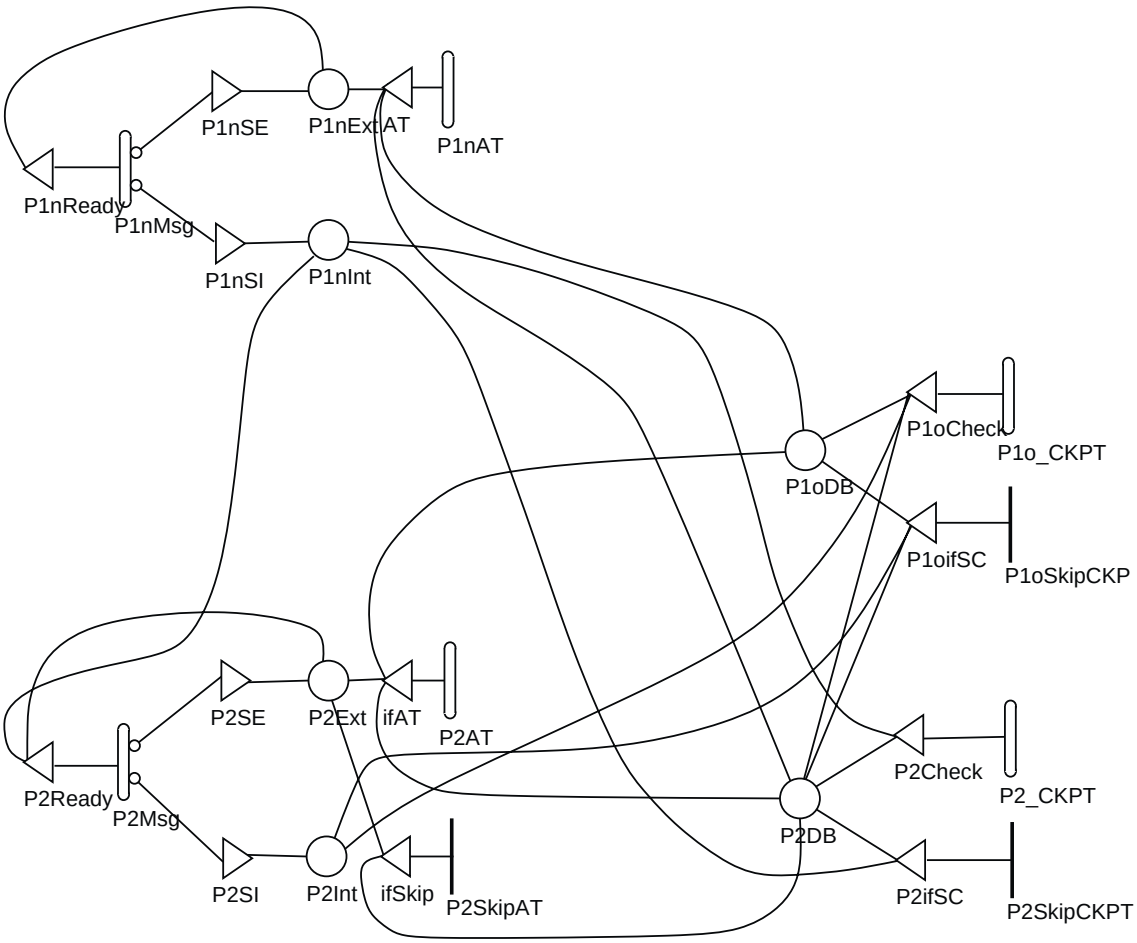


Fig. 7. SAN reward model RM_{Gp}.

In addition, due to the nature of the measures it is intended to support, the SAN reward model RM_{Gd} emphasizes the effects on system dependability of the interactions between the non-ideal environment conditions and the behavior of the MDCD protocol. Accordingly, in the model construction, we relax the design assumptions for an ideal execution environment. In contrast, the purpose of RM_{Gp} is to evaluate the performance overhead resulting from processes' error containment activities. Since those fault tolerance mechanisms are directly influenced by the design assumptions, the ideal environment assumptions are preserved in this SAN reward model.

As shown in Fig. 8, the SAN reward model RM_{Nd} is rather simple and thus is not discussed here.

5.2. SAN reward structures

As a result of model translation, each of the constituent measures is ready to be mapped to a reward structure. In addition, the *UltraSAN* tool provides us with a convenient way to define a reward structure

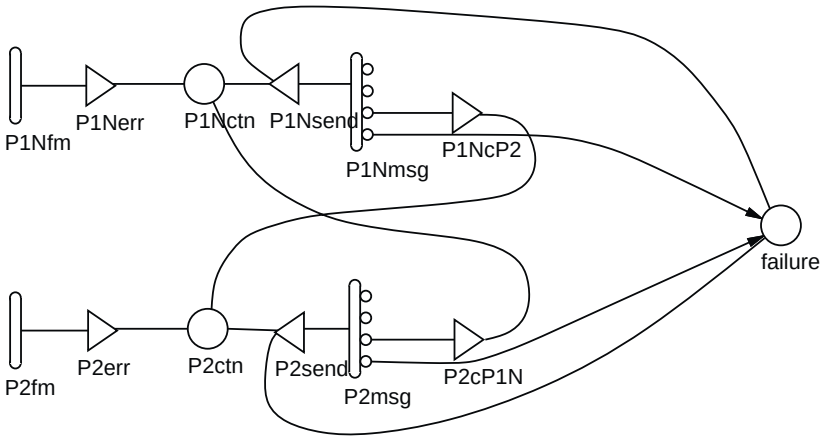


Fig. 8. SAN reward model $\mathrm{RM}_{\mathrm{Nd}}$.

by specifying a "predicate-rate" pair [13]. Below we describe how the reward structures are specified in the SAN reward models presented in the previous section.

5.2.1. Solving constituent measures in RM_{Gd}

Fig. 3 shows that the constituent measures $\int_0^\phi h(\tau)\,\mathrm{d}\tau$, $\int_0^\phi \tau h(\tau)\,\mathrm{d}\tau$, $\int_0^\phi \int_\tau^\phi h(\tau)\,f(x)\,\mathrm{d}x\,\mathrm{d}\tau$, and $P(X'_\phi \in \mathcal{A}'_1)$ are supposed to be evaluated in the reward model $\mathrm{RM}_{\mathrm{Gd}}$. Table 1 summarizes how reward structures are specified in predicate-rate pairs and how the expected reward values are computed for solving those constituent measures. An explanation of those reward model solutions has been given in Section 4.

5.2.2. Solving constituent measures in RM_{Gp}

As indicated in Fig. 3, two constituent measures are supposed to be solved in the reward model $\mathrm{RM}_{\mathrm{Gp}}$, namely, ρ_1 and ρ_2. For simplicity and clarity of the specification of the predicate-rate pairs, we instead solve for $(1-\rho_1)$ and $(1-\rho_2)$, which are the performance-overhead measures for $\mathrm{P}_1^{\mathrm{new}}$ and P_2, respectively. Table 2 enumerates the reward type and predicate-rate pair for each of the two measures.

Note that the predicate-rate pair specified for $(1-\rho_2)$ involves more conditions. This is because, unlike the process $\mathrm{P}_1^{\mathrm{new}}$ which is always considered as potentially contaminated when the system is under the G-OP mode, we dynamically adjust the confidence in P_2 and perform checkpointing and AT accordingly.

Table 1
Constituent measures and SAN reward structures in $\mathrm{RM}_{\mathrm{Gd}}$

Measure	Reward type	Predicate-rate pair	
$\int_0^\phi h(\tau)\,\mathrm{d}\tau$	Expected instant-of-time reward at ϕ	`MARK(detected)==1 && MARK(failure)==0`	1
$\int_0^\phi \tau h(\tau)\,\mathrm{d}\tau$	Expected accumulated interval-of-time reward for $[0,\phi]$	`MARK(detected)==0`	1
		`MARK(detected)==0 && MARK(failure)==1`	−1
$\int_0^\phi \int_\tau^\phi h(\tau) f(x)\,\mathrm{d}x\,\mathrm{d}\tau$	Expected instant-of-time reward at ϕ	`MARK(detected)==1 && MARK(failure)==1`	1
$P(X'_\phi \in \mathcal{A}'_1)$	Expected instant-of-time reward at ϕ	`MARK(detected)==0 && MARK(failure)==0`	1

Table 2
Constituent measures and SAN reward structures in RM_{Gp}

Measure	Reward type	Predicate-rate pair	
$1-\rho_1$	Expected instant-of-time reward at steady state	`MARK(P1nExt)==1`	1
$1-\rho_2$	Expected instant-of-time reward at steady state	`(MARK(P1nInt)==1 && MARK(P2DB) == 0) || (MARK(P2Ext)==1 && MARK(P2DB) == 1)`	1

5.2.3. Solving constituent measures in RM_{Nd}

As indicated in Fig. 3 and explained in Section 4.1, three constituent measures should be solved in the reward model RM_{Nd} (see Fig. 8), namely, $P(X''_\theta \in \mathcal{A}''_1)$, $P(X''_{\theta-\phi} \in \mathcal{A}''_1)$, and $\int_\phi^\theta f(x)\,dx$.

To solve $P(X''_\theta \in \mathcal{A}''_1)$ and $P(X''_{\theta-\phi} \in \mathcal{A}''_1)$, we assign the fault-manifestation rate of P_1^{new} to the activity that represents the fault-manifestation behavior of the first software component, and compute the expected reward values at θ and $(\theta-\phi)$, respectively. As for $\int_\phi^\theta f(x)\,dx$, since it can be interpreted as the probability that the recovered system (consisting of P_1^{old} and P_2) fails during the interval $[0, \theta-\phi]$, we assign the fault-manifestation rate of P_1^{old} to the activity that represents the fault-manifestation behavior of the first software component, and compute the complement of the expected reward value at $(\theta-\phi)$. Due to the similarity among these constituent measures, the expected instant-of-time reward values explained above can be evaluated using the same predicate-rate pair:

- *Predicate*: `MARK(failure) == 0`.
- *Rate*: `1`.

6. Evaluation results

Applying the SAN reward models described in Section 5 and using *UltraSAN*, we evaluate the performability index Y. Before we proceed to discuss the numerical results, we define the following notation:

μ_{new}	fault-manifestation rate of the process corresponding to the newly upgraded software version;
μ_{old}	fault-manifestation rate of a process corresponding to an old software version;
c	coverage of an AT;
λ	message-sending rate of a process;
p_{ext}	probability that the message a process intends to send is an external message;
α	acceptance-test completion rate;
β	checkpoint-establishment completion rate.

We begin with conducting a study of the optimality of the G-OP duration ϕ, considering the impact of the fault-manifestation rate of the upgraded software component. Specifically, we use the parameter values shown in Table 3, in which all the parameters involving time presume that time is quantified in hours. Accordingly, $\lambda = 1200$ means that the mean time between message sending events (for an individual process) is 3 s; similarly, $\alpha = 6000$ and $\beta = 6000$ imply that the mean time to the completion of an AT-based validation and the mean time to the completion of a checkpoint establishment are both

Table 3
Parameter value assignment

θ	λ	μ_{new}	μ_{old}	c	P_{ext}	α	β
10000	1200	10^{-4}	10^{-8}	0.95	0.1	6000	6000

600 ms. Further, we let γ (see Eq. (4)) be a decreasing function of $\overline{\tau}$, the mean time to error detection. More succinctly, $\gamma = 1-\overline{\tau}/\theta$. This function is defined based on the following consideration. Safeguard activities would no longer be performed after τ when error detection brings the system back to the normal mode with P_1^{old} and P_2 in mission operation; since that implies an unsuccessful (but safe) onboard upgrade, the performance cost paid for the safeguard activities up to τ would yield an additional reduction of mission worth, relative to the case of a successful onboard upgrade.

The numerical results from this study are displayed as the curve with solid dots in Fig. 9. The values of the performability index indicate that the optimal duration of the G-OP mode for this particular setting is 7000 h, which yields the best worth of the mission period θ, due to the greatest possible reduction of expected total performance degradation. This implies that for this particular setting, a ϕ smaller than 7000 would lead to a greater expected performance degradation due to the increased risk of potential design-fault-caused failure. On the other hand, if we let ϕ be larger, then the increased performance degradation due to performance overhead would more than negate the benefit from the extended guarded operation.

By decrementing the fault-manifestation rate of P_1^{new} (μ_{new}) to 0.5×10^{-4} (while letting other parameter values remain the same), we obtain another set of values of Y, as illustrated by the companion curve marked by hollow dots in Fig. 9. The two curves together reveal that the optimality of ϕ is very sensitive to the reliability of the upgraded software component. In particular, we observe that when μ_{new} is decremented

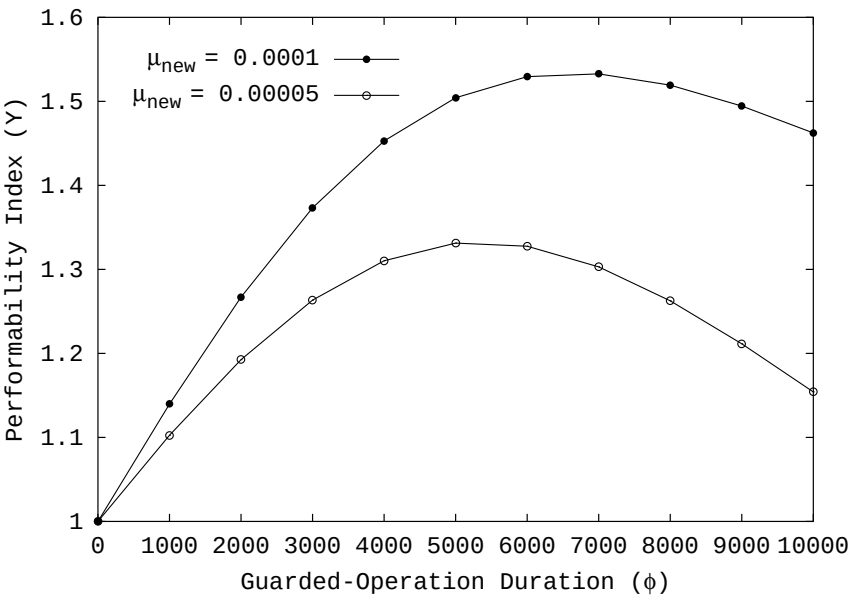


Fig. 9. Effect of fault-manifestation rate on optimal G-OP duration ($\theta = 10{,}000$).

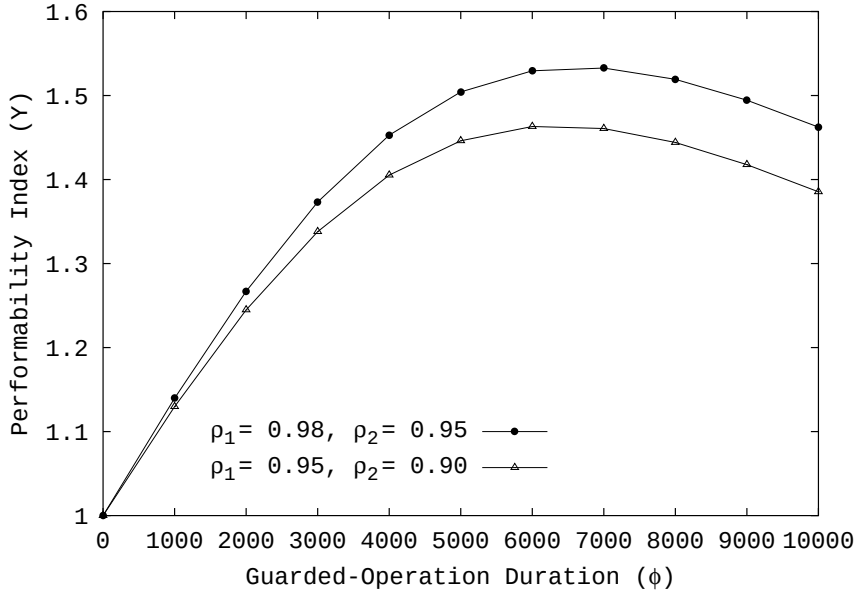


Fig. 10. Effect of performance overhead on optimal G-OP duration ($\theta = 10{,}000$).

from 10^{-4} to 0.5×10^{-4}, the optimal ϕ is dropped from 7000 to 5000 h, even though the performance costs of safeguard activities remain low (thus ρ_1 and ρ_2 remain high, and equal 0.98 and 0.95, respectively). While it is quite obvious that a smaller μ_{new} will favor a shorter duration of the G-OP mode, this study confirms the relation between the two system attributes and helps us to recognize the sensitivity of this relation.

In the next study, we change the values of α and β to 2500 (i.e., the times to completion of an AT-based validation and completion of a checkpoint establishment become 1440 ms, up from 600 ms in the previous study), implying that the performance costs for safeguard activities become higher. The evaluation results are shown as the curve with tiny hollow triangles in Fig. 10, in which we duplicate the curve with solid dots from Fig. 9 for comparison. With the decremented values of the basic parameters α and β, ρ_1 and ρ_2 (which are "derived parameters") are reduced to 0.95 and 0.90, respectively. As shown by the curve with the tiny hollow triangles, the optimal ϕ for this case is 6000, down from 7000. The change of the optimal ϕ is again a result of the tradeoffs between the two types of expected performance degradation. More specifically, the change is due to the fact that the increased performance overhead tends to further negate dependability benefits, and thus suggests an earlier cutoff line for guarded operation.

In addition, we carry out an evaluation experiment to investigate the effect of AT's coverage, c, on the optimal G-OP duration. The results are displayed in Fig. 11. Note that the top curve in the figure is duplicated from Fig. 10, and consists of the data points resulting from the evaluation experiment that assumes $c = 0.95$ and yields overhead measures equal to 0.05 and 0.10 (i.e., $\rho_1 = 0.95$ and $\rho_1 = 0.90$, respectively). We observe from the figure that while AT's coverage decreases significantly, the optimal ϕ remains the same (i.e., 6000 h), implying that the optimal value of ϕ is rather insensitive to variations of c. This indicates that the tradeoffs between the two types of performance degradation (that due to the performance cost of G-OP, and that due to potential design-fault-caused failure) chiefly involve the

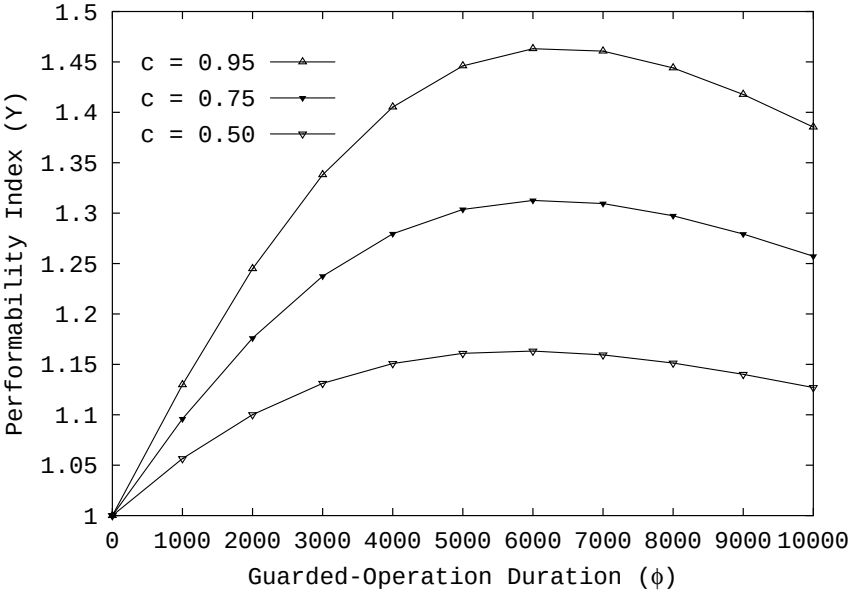


Fig. 11. Effect of AT coverage on optimal G-OP duration ($\theta = 10{,}000$).

reliability of software components and the performance overhead of safeguard activities, rather than the effectiveness of error detection, given that performance costs of AT and checkpointing are low or moderately low. On the other hand, the value of Y itself is sensitive to variations of c. As shown in the figure, the maximum value of Y drops from over 1.45 to about 1.15 when c is reduced from 0.95 to 0.50.

We have also conducted an experiment in which c is set to 0.20. In this case, the greatest value of Y equals 1.06, which is obtained when G-OP duration is 4000. However, this maximum value suggests that the gained benefit is too insignificant to justify the use of guarded operations of any lengths (when $c = 0.20$). Further, when we let c be 0.10, Y becomes less than 1 for any ϕ in $(0, \theta]$ and is a decreasing function of ϕ, implying that it is not worthwhile to consider G-OP if AT's coverage is very low. While those results are fairly intuitive, they show that: (1) the behavior of our model is reasonable, and (2) the definition of Y allows this performability measure to support decision making in various capacities. More specifically, Y not only enables us to determine the value of ϕ that would yield the greatest reduction of the expected total performance degradation, but also permits us to examine whether the amount of resulting benefit (i.e., the extent of degradation reduction) will be significant enough to justify the use of G-OP under a particular setting.

Note that so far we have used θ, the time to the next upgrade, as a constant. However, as described in Section 2, θ is chosen based on a software engineering decision (at the time onboard validation completes). The decision depends upon at least two factors: (1) the planned duty of the flight software in the forthcoming mission phases, and (2) the quality of the flight software learned through onboard validation. Accordingly, we analyze the effects of the value of θ on the optimality of ϕ. Specifically, we repeat the study that yields the results shown in Fig. 9, but reduce θ to 5000 h. The resulting curves are displayed in Fig. 12.

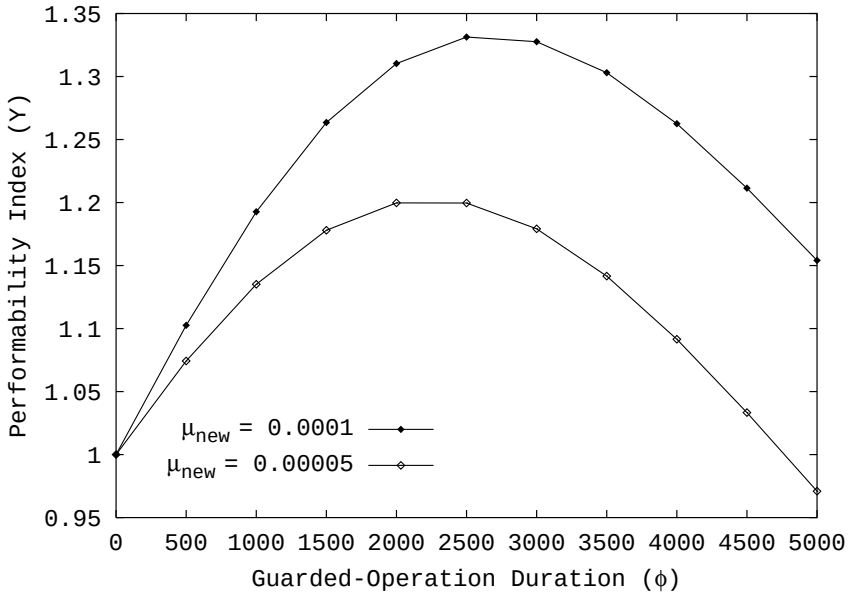


Fig. 12. Effect of fault-manifestation rate on optimal G-OP duration ($\theta = 5000$).

It is interesting to observe that while other parameter values remain the same as those shown in Table 3 (meaning that the performance and dependability attributes of the system itself are the same as in the previous study), the reduction of θ significantly changes the values for the optimal ϕ. Specifically, the optimal values of ϕ for the cases in which μ_{new} equals 10^{-4} and 0.5×10^{-4} go down to 2500 and 2000, respectively. In addition, the curves reveal that Y drops at a more significant rate soon after reaching its maximum value, relative to the case in which θ equals 10,000. This is a reasonable result because reliability is generally a decreasing function of time, for a system without maintenance. More precisely, if we view the ending point of $[0, \theta]$ as a point at which the subsequent system maintenance (i.e., the next guarded onboard upgrade) is due, then as the anticipated time to this point becomes shorter, the likelihood that the system will fail before reaching the forthcoming maintenance decreases. This, in turn, favors the decision of letting guarded operation end at an earlier point to minimize the expected total performance degradation. By inspecting the results of the constituent measures that are available to us, namely, $P(X''_\theta \in \mathcal{A}''_1)$ and $\int_0^\phi h(\tau)\,\mathrm{d}\tau$, we are able to validate this explanation.

It is worth noting that numerous analytic models for checkpointing algorithm evaluation have been developed (see [18–20] for example). The majority of checkpointing models focused attention on hardware transient faults or other types of faults that would not cause concern about the correctness of process states saved in checkpoints. For the most part, performance-dependability tradeoff studies for checkpointing algorithms were designed to help determine how often the system in question should take a checkpoint. In contrast to those models, the performability analysis conducted in this paper deals with the effects of software design faults in a distributed computing environment. More specifically, our model is concerned with uncertainties about the validity of process states and state contamination caused by message passing, and is aimed at supporting the decision on how long the system should be under protection. Our

performability measure is thereby defined in a way that takes into account system behaviors *during and beyond* the period of guarded operation. In turn, this implies that we must deal with a set of complicated sample paths and suggests the feasibility of a model-translation approach.

7. Concluding remarks

We have conducted a model-based performability study that analyzes the guarded-operation duration for onboard software upgrading. By translating a design-oriented model into an evaluation-oriented model, we are able to reach a reward model solution for performability index Y that supports the decision on the duration of guarded operation.

It is always desirable to directly apply efficient analytic techniques and existing tools for solving modeling problems. In practice, however, there are cases in which desired modeling techniques and tools cannot be immediately applied to an engineering problem we seek to solve. Although we may simplify the problem to make it fit a particular modeling method or a completely tool-based solution, we may lose important information and get results that are inaccurate or even misleading if the required simplification is excessive.

Accordingly, the intent of this investigation was not to develop a general modeling method superior to some existing techniques. Instead, our motivation has been to investigate the methods that lead to better utilization of existing modeling techniques and tools for engineering applications. As exemplified in this paper, the model-translation approach enables us to expose hidden opportunities to apply efficient model construction and solution strategies for the evaluation of an otherwise difficult performability measure. More generally, successive model translation enables us to conduct performability analyses for complex engineering applications in which boundaries and relationships between subsystems, or, between system properties considered in a performability measure, are not sufficiently clear from the original problem formulation. By promoting approaches that bridge the gap between difficult engineering applications and analytic methods developed by the research community, such as reward model solution techniques, behavioral decomposition, and hierarchical composition, this effort makes an important contribution to the area of performance and dependability modeling.

It is also worthwhile noting that unlike separate performance and dependability measures (such as response time and availability) that can be assessed directly through testbed experiments, performability measures are often defined to quantify the collective effect of various properties of a system on its "ability to perform". Accordingly, in many cases it could be very difficult for us to obtain the final solution of a performability measure directly from measurement-based or testbed-simulation-based evaluation. However, when the problem of solving a performability measure is transformed into that of evaluating constituent reward variables, it may become possible for us to choose among analytic, measurement-based, and testbed-simulation-based techniques, or a hybrid combination of them, to compute the individual measures for the final solution. Furthermore, the model-translation approach permits us to access the results of the constituent measures to gain more insight from a model-based performability evaluation.

Our current effort is directed toward continuing this investigation by carrying out more case studies in further depth. We also plan to investigate the feasibility of hybrid composition methods for evaluating constituent reward variables; for example, we might combine model-based approaches with measurement-based and/or simulation-based approaches for a performability evaluation. In addition,

since we have already developed the GSU middleware and are in the process of porting it to the Future Deliveries Testbed at JPL, we intend to experimentally validate the parameter values used in our analysis and the results of the constituent measures through applying testbed-simulation.

Acknowledgements

The authors are thankful to the anonymous reviewers to whom most revisions should be credited. The work reported in this paper was supported in part by NASA Small Business Innovation Research (SBIR) contract NAS3-99125.

References

[1] A.T. Tai, K.S. Tso, L. Alkalai, S.N. Chau, W.H. Sanders, On low-cost error containment and recovery methods for guarded software upgrading, in: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000), Taipei, Taiwan, April 2000, pp. 548–555.
[2] A.T. Tai, K.S. Tso, L. Alkalai, S.N. Chau, W.H. Sanders, On the effectiveness of a message-driven confidence-driven protocol for guarded software upgrading, Perform. Eval. 44 (2001) 211–236.
[3] A.T. Tai, K.S. Tso, L. Alkalai, S.N. Chau, W.H. Sanders, Low-cost error containment and recovery for onboard guarded software upgrading and beyond, IEEE Trans. Comput. 51 (2002) 121–137.
[4] J.F. Meyer, On evaluating the performability of degradable computing systems, IEEE Trans. Comput. C-29 (1980) 720–731.
[5] R.A. Howard, Dynamic Probabilistic Systems, vol. II, Semi-Markov and Decision Processes, Wiley, New York, 1971.
[6] W.H. Sanders, J.F. Meyer, A unified approach for specifying measures of performance, dependability, and performability, in: A. Avižienis, J.C. Laprie (Eds.), Dependable Computing for Critical Applications, Dependable Computing and Fault-tolerant Systems, vol. 4, Springer, Vienna, Austria, 1991, pp. 215–237.
[7] G. Ciardo, A. Blackmore, P.F. Chimento, J. Muppala, K.S. Trivedi, Automated generation and analysis of Markov reward models using stochastic reward nets, in: C. Meyer, R.J. Plemmons (Eds.), Linear Algebra, Markov Chains, and Queueing Models, IMA Volumes in Mathematics and Its Applications, vol. 48, Springer, Heidelberg, Germany, 1993, pp. 145–191.
[8] S. Rácz, M. Telek, Performability analysis of Markov reward models with rate and impulse reward, in: M.S.B. Plateau, W. Stewart (Eds.), Proceedings of the International Conference on Numerical Solution of Markov Chains, Zaragoza, Spain, 1999, pp. 169–187.
[9] J.B. Dugan, K.S. Trivedi, M.K. Smotherman, R.M. Geist, The hybrid automated reliability predictor, AIAA J. Guidance Contr. Dyn. 9 (3) (1986) 319–331.
[10] M. Malhotra, K.S. Trivedi, A methodology for formal expression of hierarchy in model solution, in: Proceedings of the Fifth International Workshop on Petri Nets and Performance Models, Toulouse, France, October 1993, pp. 258–267.
[11] R. Geist, Extended behavioral decomposition for estimating ultrahigh reliability, IEEE Trans. Reliab. R-40 (1991) 22–28.
[12] A.P.A. van Moorsel, B.R. Haverkort, Probabilistic evaluation for the analytical solution of large Markov chains: algorithms and tool support, Microelectron. Reliab. 36 (6) (1996) 733–755.
[13] W.H. Sanders, W.D. Obal III, M.A. Qureshi, F.K. Widjanarko, The UltraSAN modeling environment, Perform. Eval. 24 (1) (1995) 89–115.
[14] G. Ciardo, J. Muppala, K. Trivedi, SPNP: stochastic Petri net package, in: Proceedings of the International Workshop on Petri Nets and Performance Models, Kyoto, Japan, December 1989, pp. 142–151.
[15] J.F. Meyer, A. Movaghar, W.H. Sanders, Stochastic activity networks: structure, behavior, and application, in: Proceedings of the International Workshop on Timed Petri Nets, Torino, Italy, July 1985, pp. 106–115.
[16] A.T. Tai, K.S. Tso, W.H. Sanders, L. Alkalai, S.N. Chau, Low-cost flexible software fault tolerance for distributed computing, in: Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE 2001), Hong Kong, China, November 2001, pp. 148–157.
[17] B. Littlewood, D. Wright, Stopping rules for the operational testing of safety-critical software, in: Proceedings of the Digest of the 25th Annual International Symposium on Fault-tolerant Computing, Pasadena, CA, June 1995, pp. 444–453.

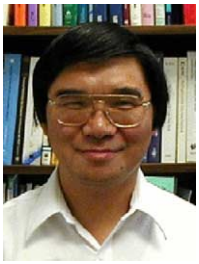
[18] V. Nicola, J. van Spanje, Comparative analysis of different models of checkpointing and recovery, IEEE Trans. Software Eng. 16 (1990) 807–821.
[19] E.G. Coffman, E.N. Gilbert, Optimal strategies for scheduling checkpoints and preventive maintenance, IEEE Trans. Reliab. R-39 (1990) 9–18.
[20] R.V. Campos, E. de Souza e Silva, Availability and performance evaluation of database systems under periodic checkpointing, in: Proceedings of the Digest of the 25th Annual International Symposium on Fault-tolerant Computing, Pasadena, CA, June 1995, pp. 269–277.

Ann T. Tai received her Ph.D. in Computer Science from the University of California, Los Angeles. She is the President and a Sr. Scientist of IA Tech, Inc., Los Angeles, CA. Prior to 1997, she was associated with SoHaR Incorporated as a Sr. Research Engineer. She was an Assistant Professor at the University of Texas at Dallas during 1993. Her current research interests concern the design, development, and evaluation of dependable computer systems, error containment and recovery algorithms for distributed computing, and distributed fault-tolerant system architectures. She authored the book, Software Performability: From Concepts to Applications, published by Kluwer Academic Publishers.

William H. Sanders received his B.S.E. in Computer Engineering (1983), his M.S.E. in Computer, Information, and Control Engineering (1985), and his Ph.D. in Computer Science and Engineering (1988) from the University of Michigan. He is currently a Professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory at the University of Illinois. He is Chair of the IEEE TC on Fault-tolerant Computing and Vice-Chair of IFIP Working Group 10.4 on Dependable Computing. In addition, he serves on the Board of Directors of ACM Sigmetrics and the Editorial Board of IEEE Transactions on Reliability. He is a Fellow of the IEEE and a Member of the IEEE Computer, Communications, and Reliability Societies, as well as the ACM, IFIP Working Group 10.4 on Dependable Computing, Sigma Xi, and Eta Kappa Nu.

Dr. Sanders's research interests include performance/dependability evaluation, dependable computing, and reliable distributed systems. He has published more than 100 technical papers in these areas. He was Co-Program Chair of the 29th International Symposium on Fault-tolerant Computing (FTCS-29), was program Co-Chair of the Sixth IFIP Working Conference on Dependable Computing for Critical Applications, and has served on the program committees of numerous conferences and workshops. He is a co-developer of three tools for assessing the performability of systems represented as stochastic activity networks: METASAN, UltraSAN, and Möbius. UltraSAN has been distributed widely to industry and academia, and licensed to more than 200 universities, several companies, and NASA for evaluating the performance, dependability, and performability of complex distributed systems. He is also a Co-Developer of the Loki distributed system fault injector and the AQuA middleware for providing dependability to distributed object-oriented applications.

Leon Alkalai is the Center Director for the Center for Integrated Space Microsystems, a center of excellence at the Jet Propulsion Laboratory, California Institute of Technology. The main focus of the center is the development of advanced microelectronics, micro-avionics, and advanced computing technologies for future deep-space highly miniaturized, autonomous, and intelligent robotic missions. He joined JPL in 1989 after receiving his Ph.D. in Computer Science from the University of California, Los Angeles. Since then, he has worked on numerous technology development tasks including advanced microelectronics miniaturization, advanced microelectronics packaging, reliable and fault-tolerant architectures. He was also one of the NASA appointed co-leads on the New Millennium Program Integrated Product Development Teams for Microelectronics Systems, a consortium of government, industry, and academia to validate technologies for future NASA missions in the 21st century.



Savio N. Chau received his Ph.D. in Computer Science from the University of California, Los Angeles. He is Principle Engineer and the Supervisor of the Advanced Concepts and Architecture Group at the Jet Propulsion Laboratory. He is currently developing scalable multi-mission avionics system architectures. He has been investigating techniques to apply low-cost commercial bus standards and off-the-shelf products in highly reliable systems such as long-life spacecraft. His research areas include scalable distributed system architecture, fault tolerance, and design-for-testability. He is a Member of Tau Beta Pi and Eta Kappa Nu.

Kam S. Tso received his Ph.D. in Computer Science from the University of California, Los Angeles, M.S. in Electronic Engineering from the Philips International Institute, Eindhoven, The Netherlands, and B.S. in Electronics from the Chinese University of Hong Kong, Hong Kong. From 1986 to 1996, he worked at the Jet Propulsion Laboratory and SoHaR Incorporated, conducting research and development on robotics systems, fault-tolerant systems, and reliable software. He is currently the Vice President of IA Tech, Inc. Dr. Tso's research interests include World Wide Web technologies, distributed planning and collaboration, high performance and dependable real-time software and systems.