Product-in-Process Performability Modeling for Guarded Software Upgrading

Ann T. Tai
IA Tech, Inc.
Los Angeles, CA 90024, USA

William H. Sanders
University of Illinois
Urbana, IL 61801, USA

1 Introduction

In his survey paper presented at the First International Workshop on Performability Modeling of Computer and Communication Systems and subsequently published in Performance Evaluation [1], Prof. John F. Meyer remarked that "it is important to note that the object of a performability evaluation study can take the form of a 'process' as well as a 'product,' e.g., a software development process, an automobile assembly process, etc. Indeed, interesting prospects for future consideration are object systems which are combinations of both." Meyer further pointed out that a performability evaluation of a "product-in-process" object system would enable us to assess the influence of process quality on product quality and/or the overall system service quality. The intent of this extended abstract is to address and foster the "product-in-process" performability evaluation by presenting an example application.

In particular, we conduct a "product-in-process" performability study for a methodology called *guarded software upgrading* (GSU) [2]. The objective of the methodology is to guard an evolvable, distributed embedded system for long-life deep-space missions against the adverse effects of design faults introduced by an onboard software upgrade. The GSU methodology is supported by a message-driven confidence-driven (MDCD) protocol that enables effective and efficient use of checkpointing and acceptance test (AT) 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, while allowing the updated component to interact freely with other components in the system. The period during which the system is under the protection of the MDCD protocol is called "guarded operation." Our (separate) model-based studies have shown that the MDCD protocol significantly improves system reliability during an onboard software upgrade and effectively reduces performance cost [3, 2]. Nonetheless, when we want to identify the optimal duration of guarded operation with respect to minimizing the expected total performance degradation (including that due to both design-fault-caused failure and the performance overhead of fault tolerance), the effects of dependability gain and performance cost must be considered jointly. Accordingly, the effort presented in this extended abstract attempts to achieve the above purpose via constructing and solving a product-in-process performability model. More precisely, the newly upgraded component in which we have not yet established enough confidence is viewed as the "product," while the guarded operation enabled by the MDCD protocol is regarded as the "process" that intends to safeguard the initial onboard use of the product. The base model is constructed using stochastic activity networks (SANs) and solved by *UltraSAN* [4].

2 Performability Variable

We aim to define a performability variable that will help us to determine how long we should apply the MDCD protocol after an upgraded software component starts its onboard execution. In other words, this interval, which we call the "duration of the guarded operation (G-OP) mode," will be determined based on the value of the performability variable that reflects the reduction of the performance degradation. We let the time between onboard software upgrades and the duration of the G-OP mode be denoted by θ and ϕ, respectively, as shown in Figure 1.

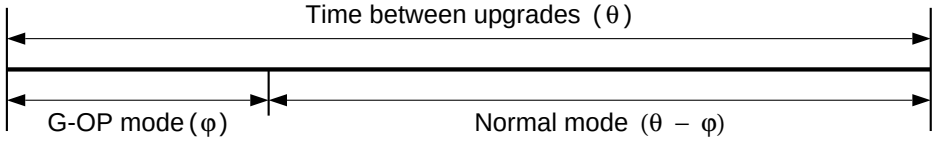


Figure 1: Duration of the Guarded-Operation Mode

As mentioned in Section 1, we consider two types of performance degradation, namely, 1) the performance degradation caused by the performance overhead of checkpoint establishment and AT-based validation, and 2) the performance degradation due to design-fault-caused failure.

Clearly, a greater value of ϕ implies decreased expected performance degradation due to potential system failure caused by residual design faults in the upgraded software component and increased performance degradation due to the overhead of checkpointing and AT. If we let D_ϕ denote the amount of total performance degradation when the duration of the G-OP mode is ϕ, then D_0 refers to the total performance degradation for the boundary case in which the G-OP mode is completely absent (having a zero duration). On the other extreme, if the performance overhead of checkpointing and AT-based validation are negligible, we may let the system be under the G-OP mode until the next upgrade. We view this extreme case as the "ideal" case in the sense that checkpointing and AT-based validation are applied throughout θ to reduce failure-caused performance degradation without introducing overhead-caused performance degradation. Accordingly, we let D_θ^{I} denote the total performance degradation for this extreme case.

It follows that a value of ϕ that makes the expected total performance degradation be closer to the expected value of D_θ^{I} can be regarded as a better choice. Since our goal is to minimize the total performance degradation via choosing an optimal ϕ, we let the performability variable take the form of a *performability index* Y, which evaluates how effectively a G-OP duration ϕ reduces the 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 $E[D_0]$ and $E[D_\theta^{\mathrm{I}}]$ to that between $E[D_\phi]$ and $E[D_\theta^{\mathrm{I}}]$:

$$Y = \frac{E[D_0] - E[D_\theta^{\mathrm{I}}]}{E[D_\phi] - E[D_\theta^{\mathrm{I}}]} \tag{1}$$

Based on the definitions of D_0, D_ϕ, and D_θ^{I} and the above discussion, we can anticipate performability benefit from a guarded operation that is characterized by a duration ϕ when $(E[D_\phi] - E[D_\theta^{\mathrm{I}}])$ is less than $(E[D_0] - E[D_\theta^{\mathrm{I}}])$. More precisely, $Y > 1$ implies that the application of guarded operation will yield performability benefit with respect to the reduction of total performance degradation. On the other hand, $Y \leq 1$ suggests that guarded operation will not be effective for total performance degradation reduction.

The development of the MDCD protocol assumes that the underlying embedded system consists of three computing nodes and two functionally different interacting application software components, one of which has two versions, namely, a better-performance less-reliable version (upgraded version) running in the foreground, and a poorer-performance more-reliable version (earlier version) running in the background to enable error recovery.

A long-life mission typically consists of critical and non-critical phases. To minimize the risk, onboard software upgrades are supposed to take place during the non-critical mission phases, during which the spacecraft does not require full computation power. Therefore, we are able to make use of a processor that otherwise would be idle, to allow concurrent execution of the new and old versions of the application software component that is undergoing an upgrade. The use of non-dedicated resource redundancy is highly desirable for a fault-tolerant avionics system. Nonetheless, it could still cause performance degradation in the sense that decreasing a processor's idle time may result in a reduction of its lifetime in deep-space environments. Accordingly, the cost of using an otherwise idle processor for fault tolerance is accounted for in the derivation of the solution for Y.

We first let ρ denote the steady-state fraction of time that a process will make "forward progress" (i.e., perform its application tasks rather than checkpointing and AT). Then, when the duration of the G-OP mode is ϕ and no failure occurs during θ, the performance degradation that is due to overhead can be expressed as $((1-\rho)2\phi+\phi)$, where the second term represents the performance cost of using the otherwise idle processor. On the other hand, if a failure occurs during θ, then the performance degradation will be the penalty from the failure and will be accounted as 3θ, since we consider the computation capacity of each of the three processors to be wasted throughout the duration θ. Accordingly, the expected value of D_ϕ can be formulated as

$$E[D_\phi] = R_\phi^{\text{MDCD}} R_{\theta-\phi}^{\text{base}}((1-\rho)2\phi+\phi) + (1 - R_\phi^{\text{MDCD}} R_{\theta-\phi}^{\text{base}})\, 3\theta \tag{2}$$

The first term evaluates the performance degradation for the failure-free case and reflects the costs of the times during which the active processes are not making forward progress and the idle processor is exploited to host the shadow process. The second term evaluates the performance degradation for the failure case and represents the penalty from wasting all three processors throughout the duration θ.

For the extreme case in which the G-OP mode is completely absent, ϕ will be equal to zero. And since R_0^{MDCD} is equal to unity, the solution for the expected value of $E[D_0]$ can be obtained from Equation (2) in a straightforward manner:

$$E[D_0] = (1 - R_\theta^{\text{base}})\, 3\theta \tag{3}$$

For the other extreme case in which the G-OP mode spans the whole duration of θ and we assume that checkpointing and AT take a negligible amount of time, ρ will be equal to one. Furthermore, since R_0^{base} is equal to unity, again we can obtain the solution of $E[D_\theta^{\text{I}}]$ from Equation (2):

$$E[D_\theta^{\text{I}}] = R_\theta^{\text{MDCD}}\, \theta + (1 - R_\theta^{\text{MDCD}})\, 3\theta \tag{4}$$

Thus, the performability index can be evaluated by solving Equations (2), (3), and (4). The values of R_t^{MDCD}, R_t^{base} and ρ are provided by the SAN submodels, which are described in the following section.

3 SAN Submodels

Although SANs' rich syntax and marking-dependent specification capability allow us to specify every aspect of the protocol precisely, the resulting state space may become unmanageable if we attempt to make the SAN model a procedural specification of the MDCD protocol. To avoid this difficulty, our approach is to minimize explicit representation of the algorithmic details, while ensuring that every aspect of their impact on the particular measure we seek to solve is captured. For example, in the SAN model for solving R_t^{MDCD}, we avoid modeling details about checkpoint establishment and rollback 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.

Likewise, in the SAN submodel for solving ρ, the performance overhead during failure-free operation, we omit those failure-behavior-related aspects, such as fault manifestation, undetected error, and dormant error conditions that remain in a process state after error recovery. Instead, we focus on representing those conditions that would require a process to take actions that do not belong to the category of "forward progress."

Due to the nature of the measure, the SAN submodel for solving R_t^{MDCD} emphasizes the effects on system reliability of the interactions between the non-ideal environment conditions and the behavior of the MDCD protocol. Accordingly, in the model construction, we relaxed the design assumptions for an ideal execution environment. In contrast, the purpose of the SAN submodel for solving ρ 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 submodel.

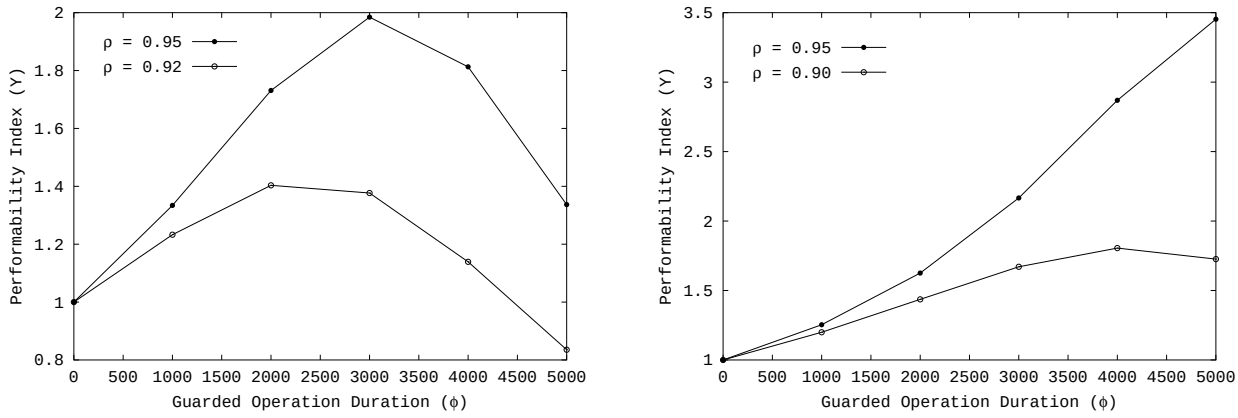
In summary, in order to prevent a state space from becoming unnecessarily large, we take a "measure-adaptive" approach in submodel construction.

4 Discussion

Using the performability index, we determine that the optimal duration of the G-OP mode. The numerical results of our first study are displayed in Figure 2(a) (the curve with solid dots). The optimal G-OP mode duration is about 3000 hours, which yields the best reduction of expected performance degradation. This implies that for this particular setting, a ϕ smaller than 3000 would lead to a greater expected performance degradation due to 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 for fault tolerance would more than negate the benefit from extended guarded operation.

In the next study, we decrease the checkpoint-establishment completion rate and AT completion rate, implying that the performance costs for checkpoint establishment and AT-based validation become higher. The evaluation results are shown by the curve with hollow dots in Figure 2(a). For this example case, the optimal duration of the G-OP mode is 2000, as revealed by the curve. This is because the increased performance overhead tends to further negate reliability benefits, and thus results in an earlier cutoff line for the guarded operation.

Figure 2(b) illustrates the impact of reliability of the upgraded software component on the optimality of ϕ. Specifically, we increase the fault manifestation rate of the upgraded



(a) (b)

Figure 2: Optimal Duration of Guarded Operation

software component (i.e., μ_{new}). Here we see that the higher μ_{new} favors a longer duration of the G-OP mode. For example, when ρ is equal to 0.90, the optimal ϕ suggested by the value of the performability index is 4000; while a greater ρ (i.e., 0.95) requires the G-OP mode to span the entire duration of θ for the maximum reduction of total performance degradation.

As mentioned in the opening section, the purpose of this study has been to investigate the feasibility of "product-in-process" performability modeling for applications such as guarded software upgrading, and to illustrate the type of results that can be obtained. The analytic results presented here suggest that the "product-in-process" performability evaluation indeed provides a means of assessing product-process interaction for degradable systems, and would be useful with regard to decision-making for various engineering processes, such as software upgrade and system maintenance, as surmised by Prof. Meyer. Indeed, we have just begun to investigate the analytic methods for "product-in-process" performability evaluation. We intend to continue our study in several respects, including the determination of criteria for defining a performability variable, methods for analyzing the interactions between a product and a process, and techniques for identifying the optimal value(s) of the parameter(s) of the process.

References

[1] J. F. Meyer, "Performability: A retrospective and some pointers to the future," *Performance Evaluation*, vol. 14, pp. 139–156, Feb. 1992.

[2] A. T. Tai, K. S. Tso, L. Alkalai, S. N. Chau, and W. H. Sanders, "Low-cost error containment and recovery for onboard guarded software upgrading and beyond," *IEEE Trans. Computers*, (To appear).

[3] A. T. Tai, K. S. Tso, L. Alkalai, S. N. Chau, and W. H. Sanders, "On the effectiveness of a message-driven confidence-driven protocol for guarded software upgrading," *Performance Evaluation*, vol. 44, pp. 211–236, Apr. 2001.

[4] W. H. Sanders, W. D. Obal II, M. A. Qureshi, and F. K. Widjanarko, "The *UltraSAN* modeling environment," *Performance Evaluation*, vol. 24, no. 1, pp. 89–115, 1995.