THREE REAL-TIME ARCHITECTURES: A STUDY USING REWARD MODELS

J. A. Sjogren
Air Force Office of Scientific Research
Washington, DC
R. M. Smith*
Yale University
New Haven, Connecticut

Abstract

Numerous applications in the area of computer system analysis can be effectively studied with Markov reward models. These models describe the evolutionary behavior of the computer system by a continuous-time Markov chain and a reward rate is associated with each state. In reliability/availability models, upstates have reward rate 1 and down states have reward rate zero associated with them. In a combined model of performance and reliability, the reward rate of a state may be the computational capacity, or a related performance measure. Steady-state expected reward rate and expected instantaneous reward rate are clearly useful measures which can be extracted from the Markov reward model. We illustrate the diversity of areas where Markov reward models may be used with a comparative study of three examples of interest to the fault tolerant computing community.

Introduction

Digital computers are playing a critical role in aviation that will only increase in the near future. The commercial Airbus 320 relies on digital control without backups. The F-18 fighter has a quadraplex fly-by-wire system, with full analog backup, and additional mechanical backup on pitch and yaw. The JAS-39 Gripen (Royal Swedish Air Force) uses triplex FBW control, with analog backup. Sooner or later, FBW technology will be declared mature enough to dispense with backups altogether, but in the meantime significant problems remain to be solved.

These issues arise from the very-high dependability requirements imposed on avionics systems, by comparison with the reputed high reliability of the aircraft structure for example. Which fault-tolerant architectures are compatible with real-time performance? How should software be certified: through testing and simulation, or by formal methods of proof?

Today I focus on one of these issues, the relationship of performance to reliability, and how to work with this relationship analytically.

A control task, such as an elevator adjustment, should be completed by its "hard deadline" or else the controlled plant may be in jeopardy. Thus if *performance* is too low, "environmental" reliability goes down. To increase performance (expressed in KIPS or MIPS) one may add new components to exploit a resulting pipelined or parallel architecture. But adding new components leads to a higher system failure rate, which lowers the "system" reliability.

Thus the performance and reliability measures are interrelated. One should also not overlook the performance burden that fault-tolerance features can place on a real-time architecture. Using the Draper Laboratories' Fault-Tolerant Processor, experimenters found a 43% degradation in throughput, due to the presence of FDIR routines (Fault-Detection, Isolation, and Reconfiguration) in its operating system.

We use the term "dynamic failure" to indicate the condition of a task failing to complete by its deadline. Many design factors affect the dynamic failure probability. Hardware capabilities must be taken into account. Also there is scheduling strategy; whether to use static (unchanging) or data-dependent *dynamic* task scheduling.

Certainly the tasks and their workload characteristics give another critical factor. The execution of a task imposes a *workload* on the processor, in terms of computation cycles needed. This workload is determined by the program's branching structure. For example, multiplication by zero might be executed more quickly than multiplication by a non-zero floating-point number. Often, the stochastic structure of a task-workload is accurately represented by a weighted-sum-of-normals distribution (which is approximated by the *discrete* distribution induced by the mean execution time of each program branch). See Figure 1.

Clearly the scheduling scheme and task execution times have an essential bearing on the probability of dynamic system failure. In general one would want to achieve a given performance level with minimal

*Supported in part by the NASA Langley Research Center under Grant NAG-1-897.

Figure 6, should be compared to the model seen in example 1. We assume that no work gets done in any of the recovery or reconfiguration states. The deadline for agreement is exactly T, the data interarrival time (since the purpose of the computer is to reliably disseminate the data to the four channels). Recall that $\mathbf{Y}_1(x,t)$ is the distribution of accumulated reward given that the system started in state 1 with no work having been done. Then

Prob. [The critical workload completes by t]

$$= \int_0^\infty \mathbf{Y}_1^c(x,t)\, \mathbf{B}'(x)\, dx.$$

Figure 7 shows how an increase in the value of λ, the failure rate, causes the dynamic failure probability, for the fixed deadline value $T = 1.0$ to increase as well.

Example 3. *Software Fault-tolerance*

A "Fault-tolerant Software" scheme is implemented on the same system as in Example 1 (subject to intermittent faults). Program Π_1 is a control routine based on a "direct method" matrix inversion, and completes as a normal distribution with mean 6 ms and Std. 2 ms, in other words N(6, 2); the answer it produces is sufficiently accurate 99.99% of the time. If Π_1 has not finished executing by time $T < 10$ ms, program Π_2 is started, which utilizes an iterative matrix algorithm and completes according to a normal $N(2, .5)$ distribution; the answer given by Π_2 will be sufficiently accurate 95% of the time. What is the probability that we have an accurate answer at 10 ms? We show the sensitivity of this probability to T.

The probability of failure is

Prob. [Π_1 does not complete correctly by T] $\times$
Prob. [Π_2 does not complete correctly in $10 - T$].

For Π_1 to complete correctly by T equals $\Phi_1(T) \times 0.9999$ where Φ_1 is the probability that Π_1 completes by T.

Similarly, for Π_2 to complete correctly in $10 - T$ equals to $\Phi_2(10 - T) \times 0.95$ where $\Phi_2(10-T)$ is the probability that Π_2 completes in $10-T$.

Thus,

$$\Phi_1(T) = \int_0^\infty \mathbf{Y}_1^c(x,T)\mathbf{B}'(x)\mathrm{d}x.$$

where $\mathbf{B}'(x)$ is the density function of $N(6,2)$, with a similar expression for $\Phi_2(10-T)$.

In Figure 8 we show several curves plotting completion probability against switch-over time; each curve corresponds to a different *variance* for the workload distribution of program Π_2. For the case shown in the dashed curve where the variance equals 0.5, we see the justification for the switch-over strategy and an optimal switch-over time of about 6.8 ms. This illustrates the value of improved understanding of software characteristics in fault-tolerant system design.

Conclusions and Future Work

We have laid some of the foundations, through the basic mathematics, the numerical solution methods and their coded implementation, as well as in modeling techniques, for the application of Markov reward models to real architectures. Future work could involve an extension to semi-Markov reward processes.

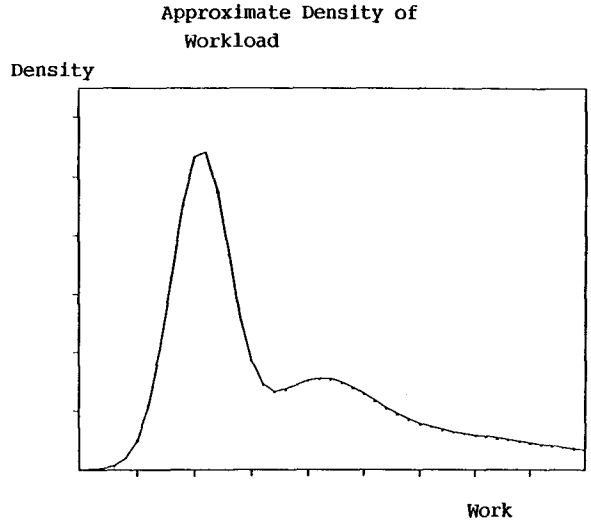


Figure 1

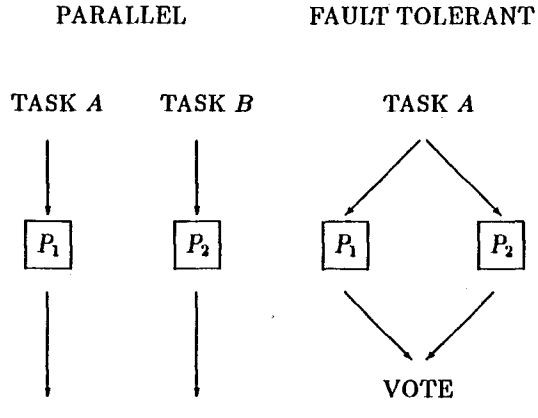


Figure 2.

unreliability. Architecture comparisons could be done with these criteria in mind. For example, in the Allied Bendix MAFT (Multicomputer for Achieving Fault Tolerance), the redundant nodes can be made to run the same schedule (as does the FTP) for greater reliability, or else to run in a distributed way as a multischedule to increase throughput. See Figure 2.

Now that we have identified some of the factors in "performability analysis", let us move on to consider a few examples, and discuss the solution techniques employed.

Example 1. *Priority Job Arrival*

Consider a single-processor system subject to intermittent faults. Faults arrive at a constant rate $\lambda = .005$, and we assume that they are instantly detected by built-in self test. Upon fault detection, the system initiates a rollback and retry procedure. The retry fails with rate γ and completes successfully with rate 10γ. We initially set $\gamma = 10.0$. Even when the system "fails", there is a global repair mechanism that completes with rate $\mu = 0.50$. The Markov model of system behavior is given in Figure 3. The times spent in states for this model are exponentially distributed. Often fault recovery has a sum-of-normals distribution, and our model would become a *semi*-Markov model.

Various reliability measures can be extracted from the model, such as the probability of the system being in the failed state at a given time t. It is natural to imagine that computational "work" is being done in state 1, but that no useful work is being done in state 2 (nor in state 3). What is the probability of x units of work having been done by time t (mission time)? An important situation occurs when a *priority job* **A** must be completed. (**A** pre-empts normal processing.) Such jobs have a fixed computational requirement of 0.75 units (CPU-seconds).

We are interested in the probability that, when a priority job **A** arrives, that it will be successfully completed. This probability depends on which state (fully functioning, retry, or failed), that the system is in when **A** arrives. We assume the system has been running for some time; therefore, steady-state probabilities accurately describe the likelihood that the system is in a given state i.

What is needed, for each of the three states, is the following information:

given this initial state, what is the probability that x units of computational work is done by a deadline (which we take as $T = 1.0$ secs)? These quantities are determined by solving a set of partial differential equations for

$$Y_i(x, 1.0) =$$

{The probability that *at most* x amount of work has been done by time 1.0, given that the starting state is i, $i = 1, 2, 3$.} This quantity is called the *accumulated reward distribution* of the model.

The new theory of Markov reward processes shows that an equivalent problem is to solve a linear system (involving symbolic variables):

$$(sI + uR - Q)\cdot \hat{Y}^*(u, s) = \mathbf{e},$$

where R is the reward structure matrix, and Q is the Markov system failure-recovery structure matrix, and the $\hat{Y}^*$ is the *double transform* of the vector $Y(x, t)$, and $\mathbf{e}$ is the "ones" column vector. See Figure 4.

The solutions we exhibit here were mainly found by a semi-symbolic inversion technique developed by R. Smith. It is an analytic method (closed-form) in the "time" variable s and numerical in the "reward" u.

Now we are ready to write down the solution to the priority job completion question. Say that **A** arrives when the system is in "retry" (state 2). Then the probability of there being sufficient computational capacity in the system to fulfill **A**'s requirement is

$$\mathbf{Y}_2^c(0.75, 1.0) = 1 - \mathbf{Y}_2(0.75, 1.0).$$

If we call this quantity w_2, then taking the other states and their steady-state probabilities into account we have the overall priority job completion probability as:

$$p_1 w_1 + p_2 w_2 + p_3 w_3 \ .$$

Some numerical answers, showing the effect of the parameter γ are shown in Figure 5.

Example 2. *Byzantine Failure*

Adherence to the fault-tolerance concept of input-data consistency requires the computer to transit the average of 4 sensor values to each of 4 computing channels. The channels perform averaging based on the spread of the values received initially. Let $D = \max[s_i] - \min[s_i]$, where $i = 1, \cdots, 4$. The number of agreement rounds required is ceiling$[D] + 1$. The increased software complexity is modeled by a workload distribution $\sim \mathbf{N}(0.02, 0.005)$, units in CPU-seconds. Experimentation gives the following discrete distribution for D:

Value	Probability	Workload
≤ 1	1/2	$N(0.02, 0.005)$
$1 < D < 2$	1/4	$N(0.04, 0.01)$
$2 < D \leq 3$	1/8	$N(.06, 0.015)$
$3 < D \leq 4$	1/16	$N(.08, 0.02)$
$4 < D$	1/16	$N(0.10, 0.025)$

The unconditional workload distribution is:

$$\mathbf{B}(x) = 1/2 \cdot N(0.02, 0.005) + 1/4 \cdot N(0.04, 0.01) + \cdots \ .$$

Even if not enough rounds have been completed by the deadline, the channels may still agree on a particular sensor value, say that held by the first channel. This is considered valid unless any one of the channels is experiencing malicious Byzantine failure, whose probability we take as $\rho = 0.001$. The underlying hardware failure model, seen in

Priority Interrupt Problem

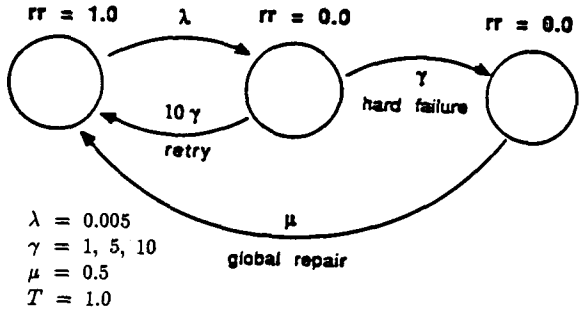


Figure 3

$$Q = \begin{vmatrix} -\lambda & \lambda & 0 \\ 10\gamma & -11\gamma & \gamma \\ 0 & \mu & -\mu \end{vmatrix} \quad R = \begin{vmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{vmatrix}$$

Figure 4

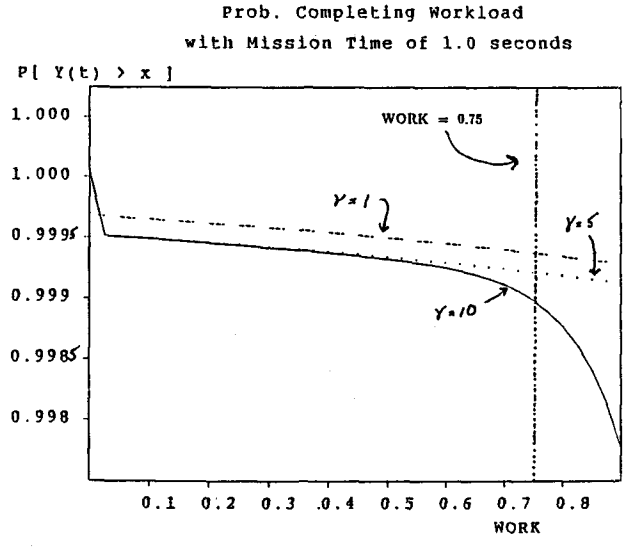
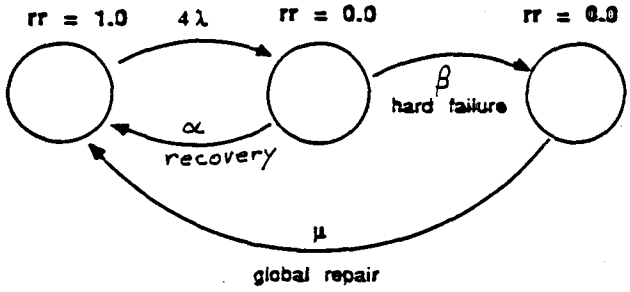


Figure 5

Hardware Failure Model

Data Dissemination Problem



λ = 0.05, 0.1, 0.2, 0.5, 1.0
α = 20.0
β = 1.0
μ = 0.5
T = 0.1

Probability of
Byzantine arrival by T $\sim$ 0.001

Figure 6

Fixed Mission Time, T = 0.1 seconds

λ	Dynamic Failure Probability
0.05	4.44×10^{-5}
0.10	4.71×10^{-5}
0.20	5.26×10^{-5}
0.50	6.88×10^{-5}
1.00	9.47×10^{-5}

Figure 7 Dynamic Failure Prob varying λ

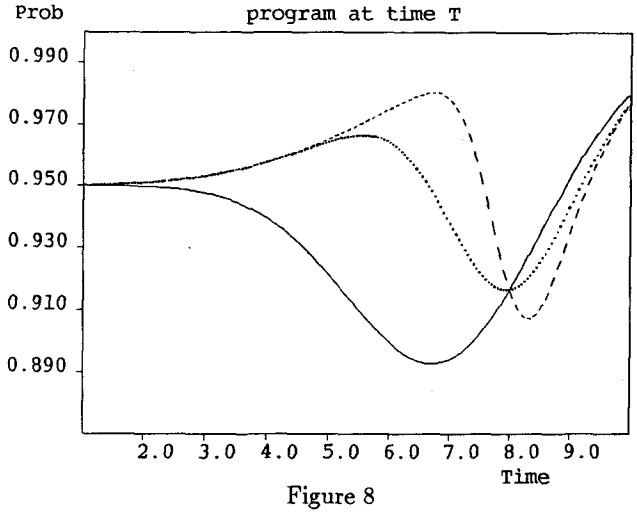


Figure 8