Performability Analysis: A New Algorithm

Hédi Nabli and Bruno Sericola

Abstract—We propose, in this paper, a new algorithm to compute the performability distribution. Its computational complexity is polynomial and it deals only with nonnegative numbers bounded by one. This important property allows us to determine truncation steps and so to improve the execution time of the algorithm.

Index Terms—Fault tolerance, repairable systems, Markov processes, performability, performance, reliability, uniformization.

1 Introduction

As recognized in a large number of studies, the quantitative evaluation of fault-tolerant computer systems requires to deal simultaneously with aspects of both performance and reliability. For this purpose, Meyer [1] developed the concept of performability of a system which is defined as the accomplishment level of the system over a specified time period t. The distribution $\mathbb{P}\{Y_t \in B\}$ is then the probability that the system performs at a level in B, where B is a set of accomplishment levels. The increasing need in evaluating performability measures comes from the fact that in highly available systems, steady state measures can be very poor, even if the mission time is not small. The use of expectations also suffers from similar drawbacks. Considering, for instance, critical applications, it is crucial for the user to ensure that the probability that its system will achieve a given performance level is high enough.

Formally, the system fault-repair behavior is assumed to be modeled by a homogeneous Markov process. Its state space is divided into disjoint subsets, which represent the different configurations of the system. A performance level (reward rate) is associated with each of these configurations. This reward rate quantifies the ability of the system to perform in the corresponding configuration. Performability is then the accumulated reward over the mission time. The distribution of this random variable (r.v.) has been studied in previous papers. Some of these papers (see [2] for references) are restricted to the case of acyclic Markov processes which are used to model nonrepairable systems.

To model the repair of faulty components in repairable systems, cyclic Markov processes are needed. For absorbing semi-Markov processes, Ciardo et al. [3] gave an algorithm to compute the distribution of accumulated reward until system failure.

The distribution of accumulated reward over a finite mission is more complex to obtain. In [4], Iyer et al. proposed an algorithm to compute recursively the moments of the accumulated reward over the mission time, with a polynomial computational complexity in the number of states. In [5], the distribution of this r.v. has been derived using Laplace transform and numerical inversion procedures to get the result in the time domain. De Souza e Silva and Gail [6] proposed a method based on the uniformization technique, however their method exibits an exponential computational complexity in the number of reward rates. Using the same technique, Donatiello and Grassi [7] obtained an algorithm with a polynomial computational complexity. However this algorithm seems to be numerically instable since the coefficient computed in their recursion can have positive and negative signs and are unbounded which can lead to severe numerical errors and overflow problems. More recently, De Souza e Silva and Gail [8] obtained also an algorithm with a polynomial computational complexity which is linear in a parameter that is smaller than the number of rewards, but their algorithm seems to have the same instability problem due the use of both positive and negative coefficients. Pattipati et al. [9] obtained the distribution of the accumulated reward for nonhomogeneous Markov processes as the solution of a system of linear hyperbolic partial differential equations which is numerically solved by using a discretization approach.

In this paper we develop a new algorithm to compute the performability distribution. As in [7] or [8], this method is based on the uniformization technique. The main contribution of this paper is that our algorithm is numerically stable by the fact it deals only with nonnegative numbers bounded by 1. Moreover, the computational complexity is improved by the use of truncation steps and the precision of the result can be given in advance. The remainder of the paper is organized as follows. In the next section, we give the proposed solution and describe the algorithm and its computational complexity. In the third section, a model of a fault-tolerant computer system is presented and solved for a given performability measure. The last section is devoted to some conclusions.

2 Model Solution and Algorithmical Aspects

We consider a system modeled by a homogeneous Markov process $X = \{X_u, u \geq 0\}$ with a finite state space $E = \{1, \ldots, M\}$. A performance level or reward rate $\rho(i)$ is associated with each state i of E. These reward rates are assumed to be time independent as usual. We denote by $r_m > r_{m-1} > \ldots > r_0$ the $m + 1$ reward rates ($m < M$). The state space E can be then divided into disjoint subsets $B_m, B_{m-1}, \ldots, B_0$ where B_i is composed by the states of E having as reward rate r_i, that is $B_i = \{j \in E / \rho(j) = r_i\}$. The process X is given by its infinitesimal generator A and by its initial probability distribution α. For any $S \subset E$, we denote by 1_S (resp. 0_S) the column vector of size the number of states in S, with all elements equal to 1 (resp. 0).

Using the uniformization technique [10], we denote by P the transition probability matrix of the uniformized Markov chain with respect to the uniformization rate λ which verifies $\lambda \geq \sup(-A(i, i), i \in E)$. The matrix P is then related to A by $P = I + A/\lambda$, where I denotes the identity matrix. In the following, to simplify notation, we will consider X as the uniformized process. For every $i, j = 0, \ldots, m$, we denote by $P_{B_iB_j}$ the submatrix of P containing the transition probabilities from states of B_i to states of B_j and by α_{B_i} the subvector of α containing the initial probabilities corresponding to states of B_i. The r.v. of interest is denoted by Y_t and represents the accumulated reward over the interval of time $[0, t]$. It is defined by

$$Y_t = \int_0^t \rho(X_u)\,du = \sum_{i=0}^{m} r_i \int_0^t \mathbf{1}_{\{X_u \in B_i\}}\,du$$

where $\mathbf{1}_{\{c\}} = 1$ if condition c is true and 0 otherwise. The r.v. Y_t takes its values in the interval $[r_0t, r_mt]$ and we wish to calculate $\mathbb{P}\{Y_t > s\}$. The reward rates r_i are arbitrary real numbers, but we assume without loss of generality that $r_0 = 0$ (see [2]). The main result of this paper, which is the distribution of Y_t is given by the following theorem (see [2] for the proof).

• *H. Nabli is with IRISA-CNRS, Campus de Beaulieu, 35042 Rennes Cédex, France.*
• *B. Sericola is with IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cédex, France. E-mail: sericola@irisa.fr.*

Manuscript received May 9, 1994.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number C95158.

THEOREM 2.1.

$$\mathbb{P}\{Y_t > s\} = \sum_{n=0}^{\infty} e^{-\lambda t}\frac{(\lambda t)^n}{n!}\sum_{k=0}^{n}\sum_{j=1}^{m}\binom{n}{k}s_j^k(1-s_j)^{n-k}b^{(j)}(n,k)1_{\{r_{j-1}t\le s<r_jt\}}$$

where

$$s_j = \frac{s - r_{j-1}t}{(r_j - r_{j-1})t}$$

and coefficients $b^{(j)}(n, k)$ are given by

$$b^{(j)}(n,k) = \sum_{l=0}^{m}\alpha_{B_l}b_{B_l}^{(j)}(n,k)$$

and column vectors $b_{B_l}^{(j)}(n, k)$ are given by the following recursive expressions

[**for** $j \le l \le m$ **and** $1 \le k \le n$]

$$b_{B_l}^{(1)}(n,0) = 1_{B_l} \text{ and } b_{B_l}^{(j)}(n,0) = b_{B_l}^{(j-1)}(n,n) \text{ for } j > 1$$

$$b_{B_l}^{(j)}(n,k) = \frac{r_l - r_j}{r_l - r_{j-1}}b_{B_l}^{(j)}(n,k-1) + \frac{r_j - r_{j-1}}{r_l - r_{j-1}}\sum_{i=0}^{m}P_{B_lB_i}b_{B_i}^{(j)}(n-1,k-1)$$

[**for** $0 \le l \le j-1$ **and** $0 \le k \le n-1$]

$$b_{B_l}^{(m)}(n,n) = 0_{B_l} \text{ and } b_{B_l}^{(j)}(n,n) = b_{B_l}^{(j+1)}(n,0) \text{ for } j < m$$

$$b_{B_l}^{(j)}(n,k) = \frac{r_{j-1} - r_l}{r_j - r_l}b_{B_l}^{(j)}(n,k+1) + \frac{r_j - r_{j-1}}{r_j - r_l}\sum_{i=0}^{m}P_{B_lB_i}b_{B_i}^{(j)}(n-1,k).$$

Note that for $j \le l \le m$ we have

$$0 \le \frac{r_l - r_j}{r_l - r_{j-1}} = 1 - \frac{r_j - r_{j-1}}{r_l - r_{j-1}} \le 1$$

for $0 \le l \le j - 1$ we have

$$0 \le \frac{r_{j-1} - r_l}{r_j - r_l} = 1 - \frac{r_j - r_{j-1}}{r_j - r_l} \le 1;$$

and for $0 \le l \le m$ we have

$$\sum_{i=0}^{m}P_{B_lB_i}1_{B_i} = 1_{B_l}.$$

For every $n \ge 0$, the initial value $b_{B_l}^{(1)}(n, 0)$ is equal to 1_{B_l} if $l \ge 1$ and the final value $b_{B_l}^{(m)}(n, n)$ is equal to 0_{B_l} if $l \le m - 1$. We then easily obtain by recurrence that $0_{B_l} \le b_{B_l}^{(j)}(n, k) \le 1_{B_l}$ Moreover, for every $j = 1, \dots, m$, and $r_{j-1}t \le s < r_jt$, we have $0 \le s_j < 1$. These remarks are essential from a computational point of view since the manipulation of nonnegative quantities bounded by 1 allows us to avoid the instability problems which may appear in the algorithms described in [7] and [8]. Let us now define for every $j = 1, \dots, m$, a partition of the state space E as

$$U_j = B_m \cup \dots \cup B_j \text{ and } D_j = B_{j-1} \cup \cdots \cup B_0.$$

For $j = 1, \dots, m$, T denoting the transpose operator, we define the following column vectors

$$b_{U_j}(n,k) = \left(b_{B_m}^{(j)}(n,k)^{\mathrm{T}}, \dots, b_{B_j}^{(j)}(n,k)^{\mathrm{T}}\right)^{\mathrm{T}}$$

and

$$b_{D_j}(n,k) = \left(b_{B_{j-1}}^{(j)}(n,k)^{\mathrm{T}}, \dots, b_{B_0}^{(j)}(n,k)^{\mathrm{T}}\right)^{\mathrm{T}}.$$

With this notation, Fig. 1 and Fig. 2 illustrate the sequence of computations (drawn only for n = 0, 1, 2, 3) that have to be done in order to evaluate the $b_{B_l}^{(j)}(n, k)$s. Note that the upper part of the diagonal of by of each triangle of cells is reported in the upper part of the first column of the next one and the lower part of the first column of each triangle of cells is reported in the lower part of the diagonal of the previous triangle of cells. The study of the recurrence described in Theorem 2.1 leads to the following remarks. In the case where $j = m$, illustrated in Fig. 2, the triangle of cells can be calculated either in a diagonal by diagonal manner provided that the first cell of a diagonal is known or in a line by line manner. In the case where $j = 1$, the triangle of cells is computed in a line by line manner but it can be also calculated in a column by column manner provided that the first cell of a column is known. This is not possible for the other triangles of cells (that is for $j = 2, \dots, m - 1$). These cells can be calculated only in a line by line manner. The way in which the computation of each cell (n, k) is performed is shown in Fig. 3. We now show that the computation of the last triangle of cells, which corresponds to $j = m$, in a diagonal by diagonal manner is very useful to reduce the complexity in the case where the value of s is near from the value of r_mt ($r_{m-1}t < s < r_mt$). Given a tolerance error ε, we define integer N as

$$N = \min\left\{n \in \mathbb{N}\,\middle|\,\sum_{j=0}^{n}e^{-\lambda t}\frac{(\lambda t)^j}{j!} \ge 1 - \frac{\varepsilon}{2}\right\}.$$

The distribution of Y_t given in Theorem 2.1 can then be written as

$$\mathbb{P}\{Y_t > s\} = \sum_{n=0}^{N}e^{-\lambda t}\frac{(\lambda t)^n}{n!}\sum_{k=0}^{n}\sum_{j=1}^{m}\binom{n}{k}s_j^k(1-s_j)^{n-k}b^{(j)}(n,k)1_{\{r_{j-1}t\le s<r_jt\}} + e(N)$$

where $e(N)$ verifies $e(N) \le \varepsilon/2$. Another truncation can be performed as follows when s is such that $r_{m-1}t < s < r_mt$. If integer C is defined as

$$C = \min\left\{c \in \mathbb{N}\,\middle|\,\sum_{h=0}^{c}e^{-x}\frac{x^h}{h!} \ge 1 - \frac{\varepsilon}{2}\right\},$$

where $x = \lambda t(1 - s_m)$, we have (see [2])

$$\mathbb{P}\{Y_t > s\} = \sum_{k=0}^{C}\sum_{n=k}^{N}e^{-\lambda t}\frac{(\lambda t)^n}{n!}\binom{n}{k}s_m^{n-k}(1-s_m)^k b^{(m)}(n,n-k) + e_1(N,C) + e(N)$$

where $e_1(N, C)$ verifies $e_1(N, C) \le \varepsilon/2$. In practice the value of s must be very close to r_mt since the requirement is generally that the r.v. Y_t is close to its maximum value r_mt with a probability close to 1; this gives a value of s_m close to 1 and thus the value of C will be small with respect to the value of N. The global computational scheme using the truncation step C is shown in Fig. 4, where only the gray part has to be computed. It is shown in [2] that the computational effort required to compute the distribution of Y_t is $O(dM[C(N - C) + mC^2/2])$ where d denotes the maximum number of nonzero entries in each row of matrix P, and the storage complexity is $O([(m - 1)C + N]M)$, where we set $C = N$ when $s \le r_{m-1}t$. The computational effort required by the method of [7] is $O(dMmN^2/2)$ and the storage requirement is also $O(mMN)$. The computational effort required by the method of [8] is stated to be $O(d\,M\,\theta N^2)$, where θ is an integer smaller than $m/2$ and equal or near to 1 in most cases and the storage requirement is $O(MN)$. Thus the algorithm proposed in this paper compares favorably with the methods of [7] and [8] when we wish to evaluate the upper tail of the distribution of Y_t, which is the case for many performability models. Another improvement in our algorithm leads to its numerical stability since it deals only with positive number bounded by 1. Some values of N and C are given in the next section.

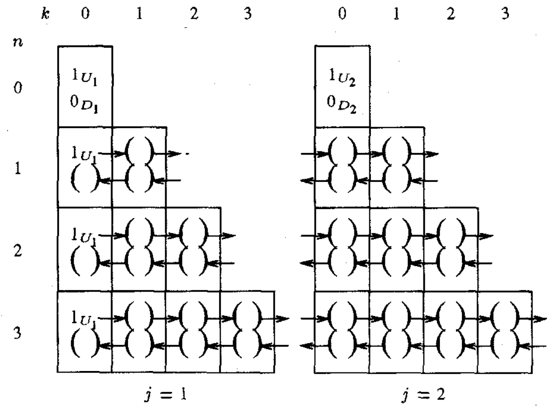
Fig. 1. In cell (n, k) the vectors $b_{U_j}(n, k)$ and $b_{D_j}(n, k)$.

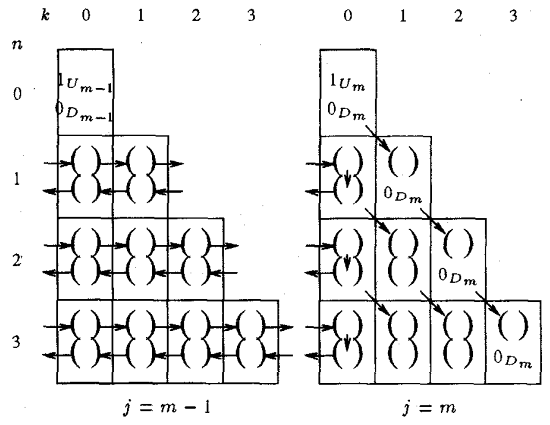
Fig. 2. In cell (n, k) the vectors, $b_{U_j}(n, k)$ and $b_{D_j}(n, k)$.

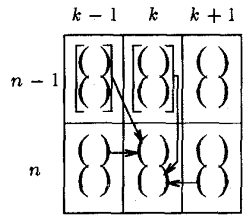
Fig. 3. Computation of cell (n, k).

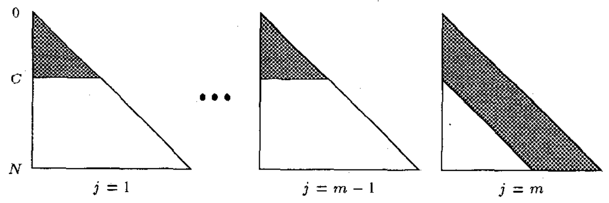
Fig. 4. In gray, the computed area.

3 Application to a Fault-Tolerant Computer System

In this section we present a model of an architecture for a fault tolerant shared memory multiprocessor which has been introduced by [11]. It consists of n CPUs, a bus and a Recoverable Shared Memory (RSM). Each CPU is composed of two processors in active redundancy whose outputs are compared in order to detect failures. Each processor gets access to the shared memory through a private cache which contains the most recent data used by the processor. This architecture has been designed in a such a way that the RSM only requires specialized hardware and can therefore use standard processors, caches and cache coherence protocols. The backward recovery protocol which is used in this architecture to tolerate some processor failures is implemented by the RSM. To provide backward recovery the basic mechanism of the RSM has to maintain two copies for each memory's location which are respectively one current copy accessible by the CPUs and one recovery copy corresponding to the previous recovery point. When a recovery point is established, the current copy is reproduced on the recovery copy so that they both contain identical data. Subsequent updates at a location only concern one single copy as the other one keeps a record of the data that was at that location at the last recovery point instant. Here we assume that both the bus and the RSM are totally reliable. On the other hand the fault which occurs in a processor can always be detected, whether it is a transient or a permanent fault, by running diagnostic checks on the defective CPU for instance. The defective CPU will still be used within the system if the fault is transient but not if permanent. The failure rate of each CPU is given by β. A transient fault occurs with probability d and a permanent one with probability $1 - d$. After such an event, the backward recovery protocol is then run during a time exponentially distributed with rate μ. Moreover, we assume that the backward recovery protocol reconfigures the system correctly with probability c and fails with probability $1 - c$. This factor c is usually called the coverage factor of the system. This establishes a Markov process shown in Fig. 5 for a number of CPU $n = 3$. State i, $1 \le i \le n$, corresponds to the state of the system where i CPU are operationnal. In this state a fault occurs with rate $i\beta$. State $n + i$, $1 \le i \le n$, corresponds to the state of the system where the backward recovery protocol tries to reconfigure the system with i CPU when the fault is transient and with $i - 1$ CPU, $i > 1$, when it is permanent. State 0 represents the situation when the system is down. The cost of fault tolerance in this architecture is mainly due to the establishment of recovery points. It has been evaluated to 30% of the power of the system in the worst case. For instance, for a standard architecture with n operational processors and no fault tolerance which has an assumed power equal to n, the reward rate r_i associated to state i, $1 \le i \le n$, is $0.7i$ in our model. The reward rates associated to the other states are null. With this reward structure associated to our model the performability distribution $\mathbb{P}\{Y_t \,/\, t > r\}$ represents the probability that the power of the architecture during $[0, t]$ averaged over time t is greater than r with r taken in the interval $[0, r_n[$.

We illustrate the model with the following parameters $c = 0.95$, $d = 0.9$ and $\mu = 1$ per second. This means for instance that the average execution time of the backward recovery protocol is 1 second. With these numerical values, Fig. 6 and Fig. 7 both show the probability that the power of the system is greater than 99.99% ($\delta = 0.9999$) of its maximum power for a one day mission time in function of the number n of CPUs and for different values of the failure rate β. We note that when $\beta = 10^{-5}$ the probability of reaching more than 99.99% of the maximum power of the system is smaller than 0.8 independently of n. We also observe that the number of

CPUs must be at most 4 if $\beta = 10^{-6}$ to obtain that probability greater than 0.95. For smaller values of the failure rate β (see Fig. 7) more than 99.99% of the maximum power of the system with a probability greater than 0.997 can only be reached if the system has two CPU's when $\beta = 10^{-7}$ and a number of CPU's equal to 8 when $\beta = 10^{-8}$. Finally, for all these computations the value of the truncation step N is $N = 87701$. The value of truncation step C increases from $C = 38$ (for $n = 2$) to $C = 109$ (for $n = 8$). These small values of C with respect to N show that the computational cost of our algorithm is, in this case, better than those presented in [7] and [8].

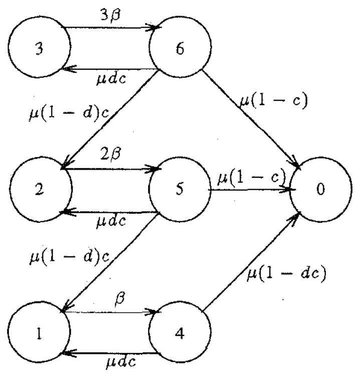

Fig. 5. The Markov process for three CPU.

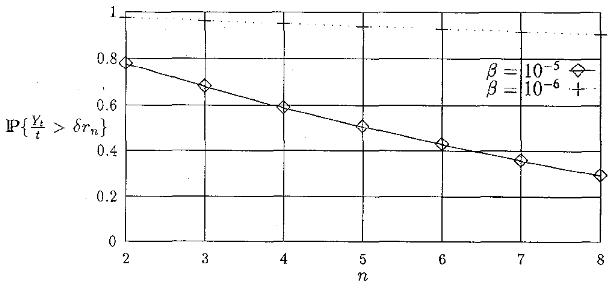

Fig. 6. A one day mission time

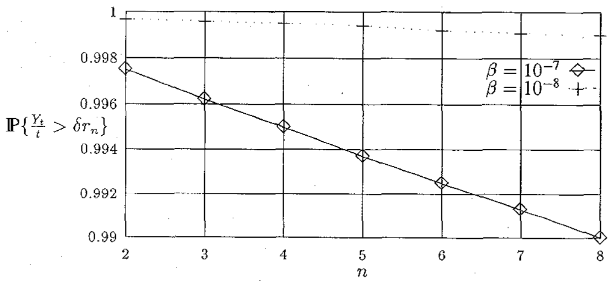

Fig. 7. A one day mission time.

4 Conclusions

The proposed method for evaluating the performability distribution leads to a new algorithm for which the number of operations is linear in the number of states of the system and linear in the number of rewards. Its main advantage with respect to existing algorithms, is that the number of operation is linear in the truncation step N and quadratic in the truncation step C which is in practice very small in comparison to N. Moreover this algorithm deals only with positive numbers bounded by 1, thus improving its stability.

Acknowledgment

This work has been partly supported by the FASST Esprit project (5212).

References

[1] J.F. Meyer, "On Evaluating the Performability of Degradable Computing Systems," *IEEE Trans. Computers*, vol. 29, no. 8, pp. 720–731, Aug. 1980.
[2] H. Nabli and B. Sericola, "Performability Analysis of Fault-Tolerant Computer Systems," Tech. Report 2254, INRIA, Campus de Beaulieu, 35042 Rennes Cédex, France, May 1994.
[3] G. Ciardo, R. Marie, B. Sericola, and K.S. Trivedi, "Performability Analysis Using Semi-Markov Reward Processes," *IEEE Trans. Computers*, vol. 39, no. 10, pp. 1,251–1,264, Oct. 1990.
[4] B.R. Iyer, L. Donatiello, and P. Heidelberger, "Analysis of Performability for Stochastic Models of Fault-Tolerant Systems," *IEEE Trans. Computers*, vol. 35, no. 10, pp. 902–907, Oct. 1986.
[5] R.M. Smith, K.S. Trivedi, and A.V. Ramesh, "Performability Analysis: Measures, an Algorithm, and a Case Study," *IEEE Trans. Computers*, vol. 37, no. 4, pp. 406–417, Apr. 1988.
[6] E. de Souza e Silva and H.R. Gail, "Calculating Availability and Performability Measures of Repairable Computer Systems Using Randomization," no. 4, *J. ACM*, vol. 36, pp. 171–193, Jan. 1989.
[7] L. Donatiello and V. Grassi, "On Evaluating the Cumulative Performance Distribution of Fault-Tolerant Computer Systems," *IEEE Trans. Computers*, vol. 40, no. 11, pp. 1,301–1,307, Nov. 1991.
[8] E. de Souza e Silva and H. R. Gail, "Calculating Transient Distributions of Cumulative Reward," Tech. Report CDS-930033, Univ. of California, Los Angeles, Sept. 1993.
[9] K.R. Pattipati, Y. Li, and H.A.P. Blom, "A Unified Framework for the Preformability Evaluation of Fault-Tolerant Computer Systems," *IEEE Trans. Computers*, vol. 42, no. 3, pp. 312–326, Mar. 1993.
[10] S.M. Ross, *Stochastic Processes*. John Wiley & Sons, 1983.
[11] M. Banâtre, A. Gefflaut, P. Joubert, P. Lee, and C. Morin, "An Architecture for Tolerating Processor Failures in Shared-Memory Multiprocessors," Tech. Report 1965, INRIA, Campus de Beaulieu, 35042 Rennes Cédex, France, Mar. 1993.