Performability Analysis of Two Approaches to Fault Tolerance

Philippe Joubert, Gerardo Rubino, Bruno Sericola

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Performability Analysis of Two Approaches to Fault Tolerance

Philippe Joubert, Gerardo Rubino and Bruno Sericola

N ° 3009

Juillet 1996

THÈME 1

ISSN 0249-6399

Performability Analysis of Two Approaches to Fault Tolerance

Philippe Joubert, Gerardo Rubino and Bruno Sericola

Thème 1 — Réseaux et systèmes
Projet Model et Solidor

Rapport de recherche n ° 3009 — Juillet 1996 — 19 pages

Abstract: We present a quantitative comparison of two popular approaches for recovering from CPU errors: Quadruple Modular Redundancy and Backward Error Recovery. Both are used in existing fault-tolerant systems offering basically the same main features and, in particular, the same fault-tolerance services (transparent recovery for hardware faults). We show that the use of performability measures is richer than classical dependability analysis. Given that they take into account not only reliability aspects but also performance metrics, they allow a deeper insight into the behaviour of the considered systems. For instance, they allow the user to identify different mission lengths leading to better adaptation of each type of architecture.

Key-words: Backward error recovery, dependability measures, fault-tolerant computing systems, performability, quadruple modular redundancy, repairable systems.

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 99 84 71 00 – Télécopie : (33) 99 84 71 71

Analyse de performabilité de deux approches de la tolérance aux fautes

Résumé : On présente une comparaison quantitative de deux approches bien connues pour le recouvrement d'erreurs CPU: la redondance modulaire quadruple et le recouvrement arrière d'erreur. Ces deux approches sont utilisées dans des systèmes existants en offrant les mêmes caractéristiques de base et, en particulier, les mêmes services de tolérance aux fautes (recouvrement transparent pour les fautes matérielles). On montre que l'utilisation de mesures de performabilité est plus riche que l'analyse classique de sûreté de fonctionnement. Etant donné qu'elles prennent en compte non seulement les aspects de fiabilité mais aussi les mesures de performance, elles permettent une analyse plus fine du comportement des systèmes considérés. Par exemple, ceci peut aider l'utilisateur à trouver une meilleure adéquation entre la durée de la mission et le type d'architecture.

Mots-clé : Recouvrement arrière d'erreur, mesures de sûreté de fonctionnement, système informatique tolérant les pannes, performabilité, redondance modulaire quadruple, systèmes réparables

1 Introduction

Fault tolerance is classically divided into four parts: error detection, fault isolation, error recovery and fault treatment [1]. The techniques for detecting and isolating faults are well understood and widely used (replication and comparison, error detecting codes ...). On the contrary, it appears that there is no clear consensus on the error recovery techniques. While all fault tolerant computers use the same techniques for error detection nearly almost each system has its own approach to error recovery. This is true even in the field of commercial fault-tolerant systems [2].

It is clear that there is no best solution to error recovery and that each error recovery approach has its own advantages and limitations. It may then be difficult for a potential customer or system designer to decide which error recovery approach is best suited to his dependability requirements. Depending on the missions assigned to the system, an approach may be well suited to a given mission type and behave poorly for another. In this area, dependability and performability modeling greatly helps the evaluation of competing design solutions in terms of implementation and performance costs. This is especially true if mission type has to be taken into account for the evaluation.

This paper presents a quantitative comparison of two different and basic approaches to CPU error recovery: Quadruple Modular Redundancy (QMR) and Backward Error Recovery (BER). Both methods are widely used in fault tolerant shared memory multiprocessors. The quantitative analysis is based upon performability measures. We propose a methodological point of view and not definitive numerical values. Our aim is to show that performability measures can be very useful in understanding the different behaviour of the two architectural choices.

The remainder of this paper is organized as follows. Section 2 describes the two error recovery techniques chosen for comparison and points out the aspects to be taken into account in the analysis. Section 3 deals with performance figures which are necessary to develop the performability analysis. Section 4 presents the models of the two architectures. We use Markov models assuming constant failure and repair rates. Section 5 proposes the dependability and performability analysis and the comparison results. The last section is devoted to some conclusions.

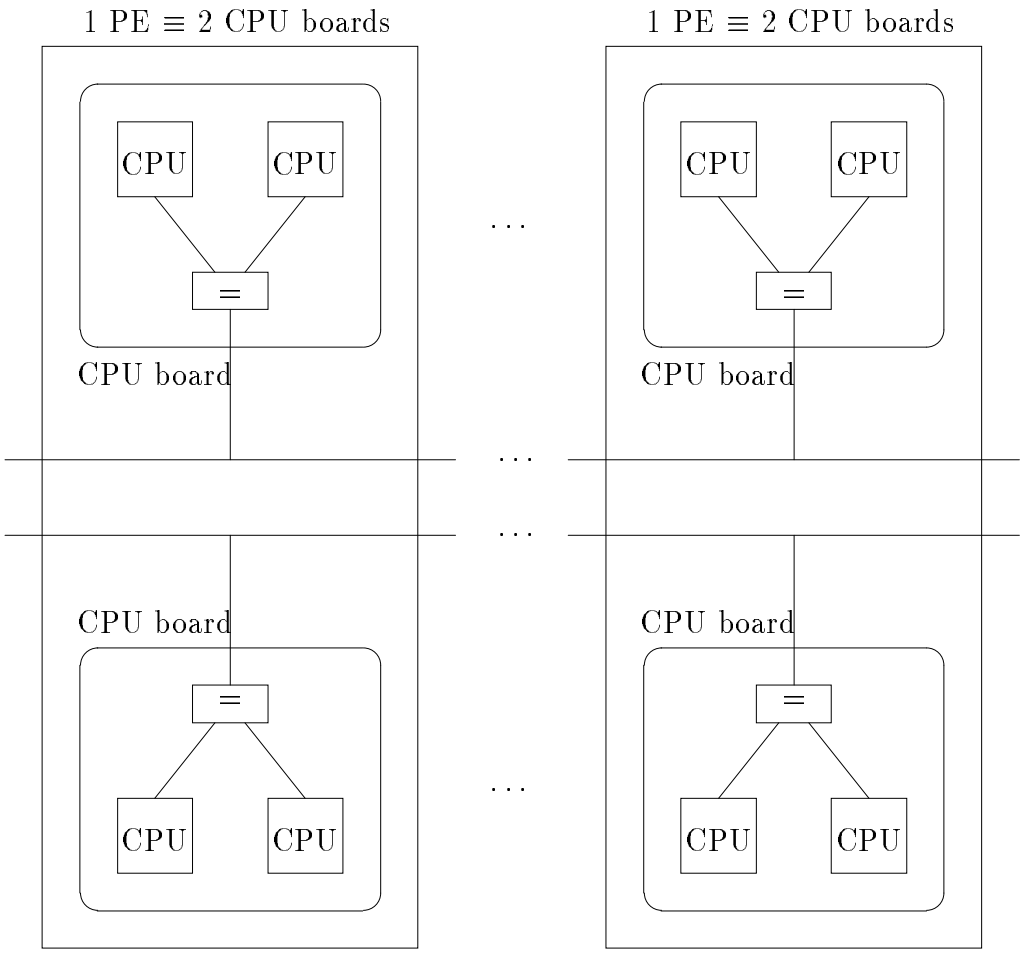
2 The QMR and BER architectures

Our concern is to compare two different alternatives for tolerating CPU faults in a fault-tolerant shared memory multiprocessor. To ease the comparison we consider two fault tolerant architectures that only differ in the way they recover from CPU errors. Apart from that, both systems are assumed to be built with the same basic units (same hardware components) and to offer the same fault-tolerance services (redundant busses, memories and IO devices).

In order to detect CPU failures, the CPU boards contain two CPUs operating in lock-step mode. The outputs of both CPUs are compared at each bus cycle. If the CPUs disagree on their outputs the board disconnects itself from the bus (fail-stop behavior). This error detection and fault isolation method is common place in fault-tolerant systems and is used for instance in the Stratus and Sequoia systems.

2.1 The QMR architecture

A first technique for recovering from CPU errors is to run the computation simultaneously on two CPU boards operating in lock-step. If one of the boards fails the other one simply continues its computation and no processing power is lost. This technique is known as Quadruple Modular Redundancy (4 CPUs are running the same computation, two on each CPU board) and is used for instance in the Stratus systems [3].



The QMR architecture

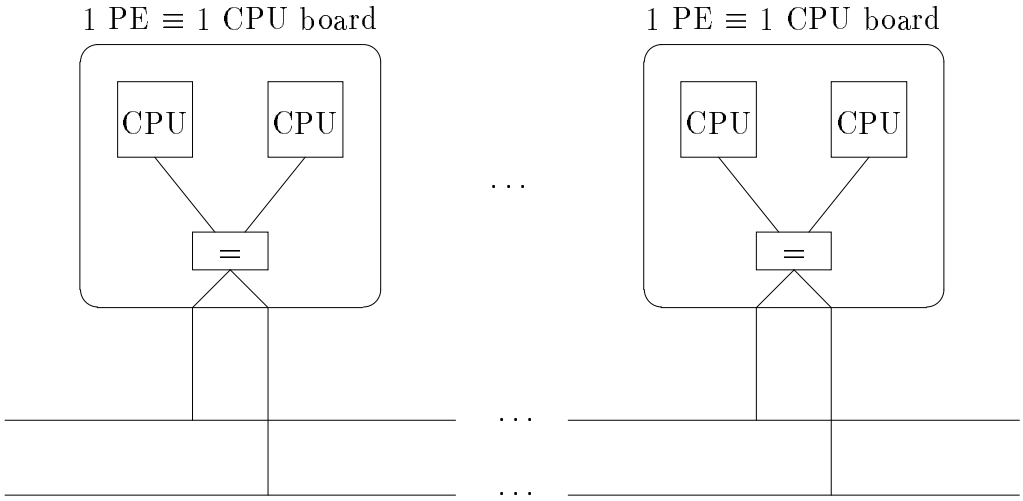
More specifically, consider the fact that one of two paired CPU boards fails. Two cases are possible:

- When the remaining board is operational, it will try to put the failed one back in operation (recall that they run exactly the same computation). In case of success, the pair of CPU boards restarts to work together, and the involved fault *was* a *transient* one. If not, the fault was *permanent*.

- If the failure of a CPU board arises when the paired one is no more operational, then the whole pair is down. Since a QMR system has no error recovery mechanism should a pair of CPU boards fail, the data that were in the caches of the paired CPU boards is lost, and the whole system is down.

2.2 The BER architecture

Another alternative is to use Backward Error Recovery (BER). The basic principle of backward error recovery is to periodically record recovery data in order to recover the state of the system to a prior state which is assumed to be error free, should an error occur. In a shared memory multiprocessor, shared memory is a natural location for storing the recovery data. One requirement is that the update of the recovery data (i.e. checkpointing) needs to be atomic (i.e. *all or nothing property*) with respect to CPU failures. This can be achieved by keeping the data modified since the last checkpoint within the cache associated with each CPU. When a checkpoint is required, the cache is flushed to the shared memory along with the CPU registers. To enforce the atomicity of cache flushes, the cache is actually flushed to two distinct memory boards.



The BER architecture

When a CPU board fails, all the processing done since the last checkpoint is lost. The shared memory contains the state of the system at the time of the last checkpoint.

If the fault was deemed to be transient, the computation simply restarts from the last checkpoint. If the fault is permanent then that CPU board can not be used further, and its computation is restarted on another board.

The above BER technique is similar to that used by Sequoia [4]. Other proposals for fault tolerant shared memory multiprocessors exist in the literature [5]. They mainly focus on reducing the performance overhead of checkpointing by using different cache flushing and memory update schemes. They also differ in the algorithms used to enforce checkpoint consistency when multiple CPUs share memory.

2.3 Comments

The main advantage of QMR is that it masks the first CPU permanent fault appearing in a paired sets of boards. Moreover, its implementation is much more simpler than in the case of a BER. On the other hand, half of the potentially available processing power of the system is unused.

The advantages and drawbacks of BER are nearly the opposite: all the CPU boards in the system are effectively used but at the expense of some processing power (checkpointing operations and recovery procedures). Moreover, the processing power of the system diminishes as CPU boards fail since the load of the failed boards has to be shared between the remaining ones. Another interesting feature of backward error recovery is its ability to tolerate some transient software faults. If a software fault is activated, for instance, by a marginal synchronization condition and the backward error recovery procedure is triggered, the computation is rolled back. The execution environment may then have sufficiently changed so that the fault is not activated anymore. This feature of backward error recovery has proved its efficiency in commercial fault-tolerant systems [6].

A third alternative for error recovery is to use Triple Modular Redundancy (TMR), that is, triplicated CPUs voting on their outputs as in the Tandem Integrity [7]. This interesting alternative is not discussed in our paper.

2.4 Processing Element (PE) concept

To avoid confusion we will focus in the following on Processing Elements (PEs) rather than on CPUs or CPU boards. A PE is, by definition, the atomic unit of processing in both architectures. It is equivalent to one CPU board in the BER architecture and to two (paired) CPU boards in the QMR one. Two multiprocessor systems having N PEs

(alive), the first using QMR, the second using BER, offer the same logical power to the user. Resuming, we have

> **QMR case.** The N PEs are built using N paired CPU boards, that is, $2N$ CPU boards (i.e., a total of $4N$ CPUs). Either all the PEs are alive (possibly with only one active CPU board), or the whole system is down.
>
> **BER case.** The N PEs are built using N CPU boards (i.e., a total of $2N$ CPUs). The system can degrade gracefully from N to only one operational PE.

3 Performance evaluation

Consider two architectures, both with N initial PEs, one using QMR, the second using BER. As in multiprocessor systems, there is an overhead which means that the *power* (given in "equivalent PEs per unit of time"), when there are N initially working PEs, is less than N. Let us denote the respective powers by p_N^{QMR} and p_N^{BER}. In the latter case, if initially the system had N PEs and at some point in time there are n operational PEs, the power is p_n^{BER}.

The computation of the power measure is a complex task. The overhead due to the multiprocessing involves bus contention, memory access delays, cache coherence protocol costs, etc. In the BER case, we have supplementary difficulties since the protocol itself adds overheads that depend on several correlated factors:

- The checkpointing rate which is determined by the behavior of the application running on the system (external IO rate, cache behavior, bounded rollback time).
- The amount of data that needs to be flushed from the PE's cache when a checkpoint occurs. This is determined by the checkpointing rate and the behavior of the application (ie. how frequently the application modifies distinct memory locations).
- The current load of the shared bus. Cache flushes use bus bandwidth that is not available to other PEs for performing useful work. This may saturate the bus and severely degrade performance.

For these reasons the values of the power that we used in the numerical tests are averages extracted from previous simulation experiments that were conducted in order to estimate

the performance of alternative checkpointing schemes for fault tolerant shared memory multiprocessors [8]. Not surprisingly, the simulations show that the overhead increases with the number of PEs, and that it is obviously more important in the BER case. The values were obtained by averaging the simulation results obtained for a set of scientific parallel applications. Of course, they may not be completely representative of the behavior of the application running on the system but they allow to perform some trend analysis.

To illustrate the paper, we use the following data sets, compacted as power values. In the case of a QMR, we have

j	1	2	3	4	5	6	7	8
p_j^{QMR}	0.90	1.25	2.55	3.35	4.07	4.72	5.37	5.92

For a BER under the same conditions, we have

j	1	2	3	4	5	6	7	8
p_j^{BER}	0.86	1.19	2.40	3.02	3.62	4.05	4.55	4.97

These results show the performance overhead of the BER architecture.

4 Models for dependability and performability analysis

We assume that each CPU board fails with a constant failure rate denoted by δ. This means that the failure rate of a PE in the QMR architecture is 2δ while its value is δ in the BER case. We assume that the probability that any given fault is a *transient* one is constant and we denote it by d. In the analysis, we will denote the common number of PEs initially working in both architectures by N.

The life-time of the system is defined here as the interval separating the starting instant from the point in time where no more self-reparing procedure is possible and an external action is necessary to put the system back to operation. In the QMR case, this happens when the first PE fails. Such an architecture cannot recover when a PE is completely down. In the BER architecture the system is down when there is no more operational PE or when the recovery protocol fails. The success of the protocol is assumed to occur with a fixed probability denoted by c, the *coverage factor*.

4.1 The QMR model

Recall that, in order to make the comparison between the two architectures easier to understand, we assume that everything but the processing units (PEs) is identical in both systems and that all these components are fail-free. This is of course false in reality but our purpose is to concentrate our analysis on the CPU error recovery procedures. The study of the effects of the rest of the components (busses, IO devices, etc.) on the behavior of the two systems, even if these components are identical, is a topic of interest in itself and not developed here.

We use a model having $2N+2$ states denoted by u_0, u_1, ..., u_N (which are the operational states), h_1, ..., h_N and an absorbing state 0. State u_n means that the system is up and that there are n "complete" PEs, that is, n PEs having their two paired boards operational. Thus, in state u_n there are $N-n$ PEs working with only one CPU board ($N-n$ "incomplete" PEs). When the system is in the absorbing state 0, it is assumed to be down and no internal action can put it back to operation. When the model is in state u_n, the failure of any CPU board of the $N-n$ incomplete PEs puts it in the absorbing state 0. This happens with rate $(N-n)\delta$. Always when the model is in state u_n, if one of the boards in the n complete PEs fails, the model goes to state h_n, with rate $2n\delta$. State h_n represents the system trying to put the failed board back into operation. From state h_n, two transitions are possible, depending on the type of the concerned fault, transient (probability d) or permanent (probability $1-d$): denoting by μ' the rate of the recovering action, with rate $\mu' d$ the model goes back to u_n and with rate $\mu'(1-d)$, that is, in the case of a permanent fault, there is one less complete PE and the new state is u_{n-1}.

With these assumptions, we obtain the Markov process depicted in Figure 1 where to simplify the picture, we show the case of $N=3$ PEs. In the scheme, the arrows without destination go to the absorbing state 0.

4.2 The BER model

The model of a system based on BER has $2N+1$ states where N are operational, again denoted by u_1, ..., u_N. State u_n means that there are n operational PEs (and, thus, that $N-n$ PEs have failed). When in state u_n, after a failure the model goes to state h_n (associated transition rate $n\delta$) representing the fact that the BER protocol is in progress. From state h_n, in case of a transient fault and of a success of the recovery protocol, the model jumps back to state u_n. The corresponding transition rate is μcd where μ is the rate of execution of the protocol. If the fault is permanent and the recovery procedure

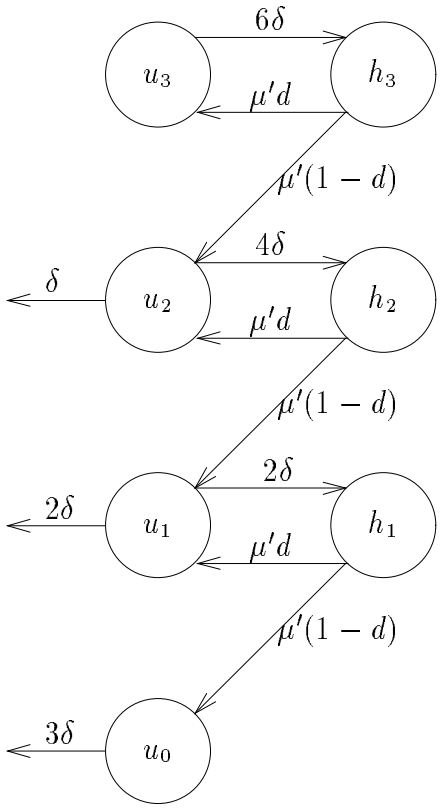


Figure 1: QMR model illustrated for 3 PEs

succeeds, the model goes to state u_{n-1} since the faulty PE is no more available. In all cases (transient or permanent fault), if the recovery protocol can not put the system back to operation, the model jumps to the only absorbing state 0 (rate $\mu(1-c)$).

With these assumptions, we obtain the Markov process depicted in Figure 2, again for 3 PEs, where the arrows without destination go to the absorbing state 0.

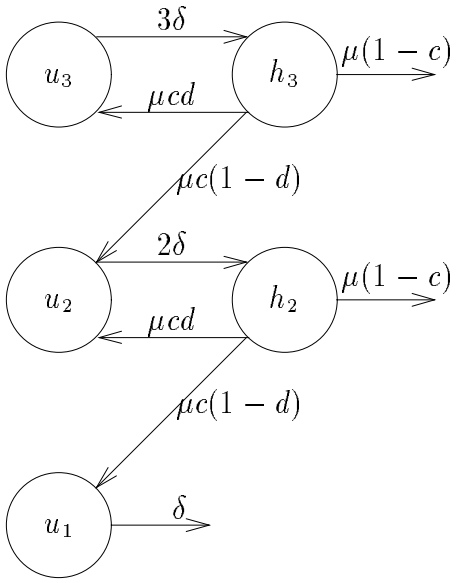


Figure 2: BER model illustrated for 3 PEs

5 Dependability and performability mean value analysis

The two architectures are compared via two expected measures. The first one is the classical Mean Time To Failure (MTTF). The second one, which depends on the mission time $(0,t)$ during which the system is supposed to be used, is the mean processing power over the interval $(0,t)$.

More formally, let $X = \{X_u,\ u \geq 0\}$ be the Markov process modeling one of the two architectures. The mean time to failure is then defined as $\mathbb{E}(T)$, where T is the life-time of X, that is, $T = \inf\{u | X_u = 0\}$.

It is shown in the Appendix that for the QMR architecture with N PEs, the MTTF is equal to x_N which is recursively given, for $j \geq 1$, by

$$x_j = \frac{1}{N-(2d-1)j}\left[\frac{1}{\delta} + \frac{2j}{\mu'} + 2j(1-d)x_{j-1}\right], \quad x_0 = \frac{1}{N\delta}.$$

It is also shown in the Appendix that for the BER architecture with N PEs, the MTTF is equal to y_N which is recursively given, for $j \geq 2$, by

$$y_j = \frac{1}{1-cd}\left[\frac{1}{j\delta} + \frac{1}{\mu} + c(1-d)y_{j-1}\right], \quad y_1 = \frac{1}{\delta}.$$

With each state x in the models, we associate a reward r_x given by $r_{u_n} = p_n$, $r_{h_n} = 0$ and $r_0 = 0$. We have $p_n = p_n^{\mathrm{QMR}}$ for the QMR model and $p_n = p_n^{\mathrm{BER}}$ for the BER model. The mean processing power over the interval $(0,t)$ is $\mathbb{E}(Y_t)$, where Y_t is the random variable defined by

$$Y_t = \frac{1}{t}\int_0^t r_{X_s} ds.$$

It follows that, for N PEs, $\mathbb{E}(Y_t)$ is given by

$$\mathbb{E}(Y_t) = \frac{1}{t}\sum_{u_n \in U} p_n \int_0^t \mathbb{P}(X_s = u_n) ds.$$

where U denotes the set of operational states.

The random variable Y_t has been studied in numerous papers in order to compute its distribution (also called *performability*) or its moments. Even its expectation is difficult to obtain in closed form and thus numerical algorithms must be used to compute it [9, 10, 11]. We use here an algorithm that has been developed especially for stiff models [12].

In our numerical evaluations, we have chosen the following data values. The CPU board failure rate is $\delta = 10^{-6}$ in failures per second. The probability that a failure is transient is $d = 0.9$.

Let us analyze first the MTTF of the two systems as a function of the number N of PEs varying from 2 to 16. In the case of the QMR architecture, the "repair" rate is $\mu' = 10^3$ in actions per second. The number $1/\mu'$ can be seen as the mean time needed to determine the nature of the fault (transient or permanent) affecting a CPU board. In the BER case, we have a repair rate of $\mu = 1$ action per sec; $1/\mu$ represents the mean execution time of the recovery protocol [4]. Last, we consider a coverage factor of $c = 0.99$.

The evolution of the MTTFs when the number of PEs varies is shown in Figure 3.

This figure shows that in the case of the QMR architecture, the MTTF is a decreasing function of the number of PEs while for BER, there is a maximal MTTF value for a 8 PE configuration. The fact that the MTTF decreases with N is a consequence of the fact that as PEs are added to the system, the total failure rate increases, while on the other side

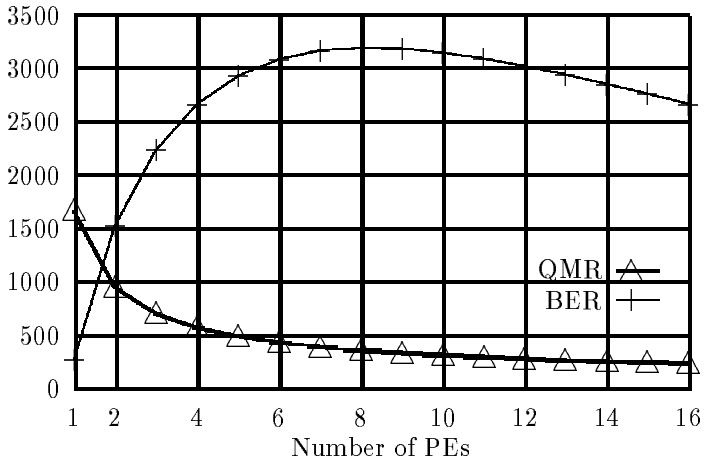


Figure 3: MTTF of both architectures in hours

the repair "device" (the backward error recovery with its coverage factor) remains alone against the failure "generator". The numerical values chosen for our experiences are such that for small values of N the MTTF increases when we add new units. For other input values (in particular, for smaller values of the coverage factor c), curves like the QMR one are also obtained in the BER case.

Clearly, this criterion is not enough to choose between the two systems. Of course, if the MTTF is the only quantitative parameter to be taken into account, the user will look at the MTTFs against the cost of both systems. The comparison depends, with our assumptions, basically on the fact that QMR needs twice the number of CPU boards while the BER needs a more sophisticated recovery protocol. Since we are only concerned here with the quantitative aspects of the considered architectures, the point is that it is not enough to know how much time the system can live without external actions. It is also important to know *how much work* it will be able to perform and for a given *mission time*, that is, over a given mission period $(0, t)$.

In Figure 4 we plot the expectation of Y_t against t which varies from 0 to 20 $\times 10^6$ secs, for a 2–PEs configuration. It is easy to show and to understand that $lim_{t\to\infty} \mathbb{E}(Y_t) = 0$ in all cases.

This curve says that for two systems offering the same maximal power equivalent to 2 PEs, the BER one is better except for short mission lengths. In Figure 5 the same metrics are shown for 8 PEs, and the same conclusion holds.

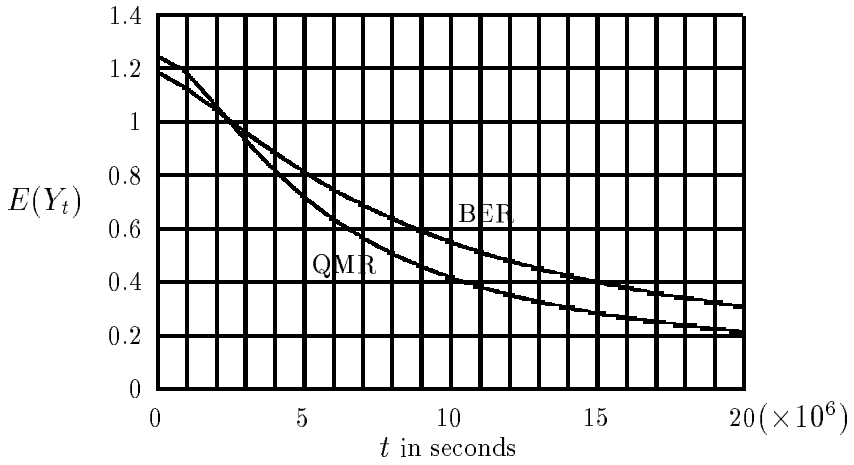


Figure 4: $\mathbb{E}(Y_t)$ for 2 PEs, as a function of the mission time t

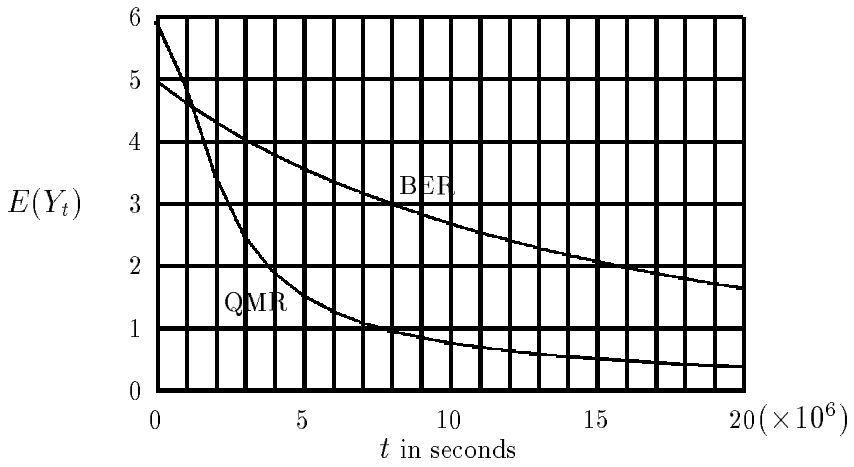


Figure 5: $\mathbb{E}(Y_t)$ for 8 PEs, as a function of the mission time t

In the next figure (Figure 6), we plot the value of the intersection point as a function of the number of PEs. The resulting curve divides the plane in two areas. The area over the curve gives the mission lengths for which BER is more efficient. Under the curve we have the points (number of PEs, mission length) where QMR is better. Our claim is that this type of analysis can help the choice between these two systems.

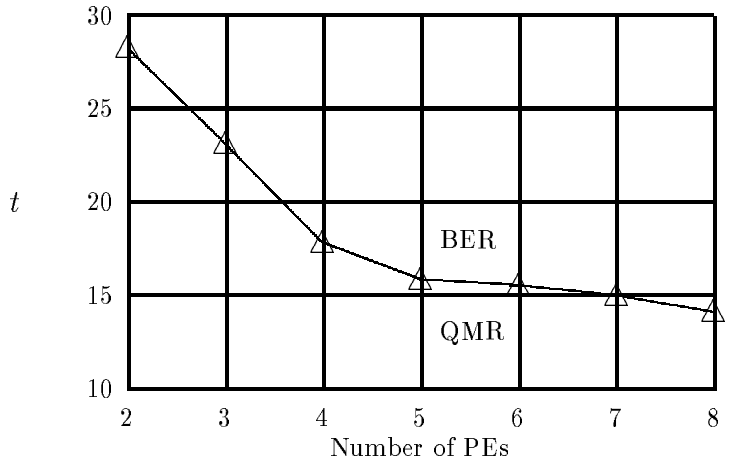


Figure 6: Choice between QMR and BER, t in days

6 Conclusions

In this paper, we develop a quantitative comparison of two important solutions to the problem of fault tolerance in multiprocessor systems with common shared memory. A first goal of the paper is to point out the interest in complementing classical reliability analysis with more sophisticated transient measures taking into account the behavior of the systems during previously specified mission periods. A second result is the specific comparison between the two architectures, QMR and BER, which allows to give numerical support to some intuitive ideas. Basically, we show how the approach followed here can identify (and quantitatively evaluate) the mission lengths where one of the two fault-tolerant solutions is better than the other. The analysis presented here can be adapted to other models, of course. It can also be complemented by the use of other measures based on the same input data, as for instance the mean total work done by the systems up to the external action necessary to restart them.

Appendix

We show in this section how the expressions for the mean time to failure (MTTF) are obtained for both architectures.

MTTF for the QMR architecture

Consider the QMR model depicted in Figure 1 with N processing elements. Let (X_t) denote the corresponding Markov process and T the life-time of the process. The mean time to failure is then equal to $\mathbb{E}(T/X_0 = u_N)$, since the initial state is state u_N.

Let us define x_j by $x_j = \mathbb{E}(T/X_0 = u_j)$ and z_j by $z_j = \mathbb{E}(T/X_0 = h_j)$. We then get, by the use of the Markov property, the following recurrence relations satisfied by the x_j's and the z_j's, for $j = 1, \ldots, N$:

$$\begin{aligned} x_j &= \frac{1}{(N+j)\delta} + \frac{2j}{N+j} z_j \\ z_j &= \frac{1}{\mu'} + d x_j + (1-d) x_{j-1} \end{aligned}$$

and $x_0 = 1/N\delta$.

By replacing the expression of z_j in the expression of x_j, we get, for $j > 0$,

$$x_j = \frac{1}{N - (2d-1)j} \left[\frac{1}{\delta} + \frac{2j}{\mu'} + 2j(1-d)x_{j-1} \right].$$

MTTF for the BER architecture

In the same way, consider now the BER model depicted in Figure 2 with N processing elements. Let (X_t) denote the corresponding Markov process and T its life-time. The mean time to failure is then equal to $\mathbb{E}(T/X_0 = u_N)$, since the initial state is state u_N.

Let us define y_j by $y_j = \mathbb{E}(T/X_0 = u_j)$ and z_j by $z_j = \mathbb{E}(T/X_0 = h_j)$. We then get, by the use of the Markov property, the following recurrence relations satisfied by the y_j's and the z_j's, for $j = 2, \ldots, N$:

$$\begin{aligned} y_j &= \frac{1}{j\delta} + z_j, \\ z_j &= \frac{1}{\mu} + cdy_j + c(1-d)y_{j-1}, \end{aligned}$$

and for $j = 1$, we get $y_1 = 1/\delta$.

By replacing the expressions of z_j in the expression of y_j, we get, for $j > 1$,

$$y_j = \frac{1}{1-cd} \left[\frac{1}{j\delta} + \frac{1}{\mu} + c(1-d)y_{j-1} \right].$$

References

[1] P.A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*, volume 3 of *Dependable Computing and Fault-Tolerant Systems*. Springer Verlag, second revised edition, 1990.

[2] J.J. Stiffler. Fault-tolerant architectures – past, present and (?) future. In M. Banâtre and P. A. Lee, editors, *Hardware and Software Architectures for Fault Tolerance. Experiences and Perspectives*, number 774 in LNCS, pages 117–121. Springer-Verlag, 1994.

[3] S. Weber and J. Beirne. The stratus architecture. In *Proc. of 21st International Symposium on Fault-Tolerant Computing Systems*, pages 79–85, Montréal, Canada, June 1991.

[4] Ph. A. Bernstein. Sequoia: A fault-tolerant tightly coupled multiprocessor for transaction processing. *IEEE Computer*, 21(2):37–45, February 1988.

[5] K.L. Wu, W.K. Fuchs, and J.H. Patel. Error recovery in shared memory multiprocessors using private caches. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):231–240, April 1990.

[6] I. Lee and R.K. Iyer. Faults, symptoms and software fault tolerance in the tandem guardian90 operating system. In *Proc. of 23rd International Symposium on Fault-Tolerant Computing Systems*, pages 20–29, Toulouse, June 1993.

[7] D. Jewett. Integrity s2: A fault-tolerant unix platform. In *Proc. of 21st International Symposium on Fault-Tolerant Computing Systems*, pages 512–519, Montréal, Canada, June 1991.

[8] M. Banâtre, A. Gefflaut, P. Joubert, P.A. Lee, and C. Morin. An architecture for tolerating processor failures in shared-memory multiprocessors. Research report 485, University of Newcastle upon Tyne, July 1994. To appear in IEEE Trans. Comp.

[9] H. Nabli and B. Sericola. Performability analysis: A new algorithm. *IEEE Trans. Comp.*, 45(4):491–494, April 1996.

[10] E. de Souza e Silva and H. R. Gail. Calculating transient distributions of cumulative reward. Technical Report CDS-930033, UCLA, Los Angles, USA, September 1993.

[11] A. Reibman, R. Smith, and K. Trivedi. Markov and Markov reward model transient analysis: An overview of numerical approaches. *European Journal of Operational Research*, 40:257–267, 1989.

[12] H. Abdallah, R. Marie, and B. Sericola. Computation of the expected interval availability for stiff Markov models. In *7th European Simulation Symposium*, pages 393–396, Erlangen - Nuremberg, October 1995. SCS.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399