Evaluation of Fault-Tolerant Multiprocessor Systems for High Assurance Applications

F. Grandoni**, S. Chiaradonna*, F. Di Giandomenico** and A. Bondavalli***

* IEI/CNR, Via V. Alfieri 1, 56010 Ghezzano, Pisa, Italy,
Email (digiandomenico, grandoni)@iei.pi.cnr.it

* * CNUCE/CNR, Via V. Alfieri 1, 56010 Ghezzano, Pisa, Italy,
E-mail S.Chiaradonna@cnuce.cnr.it

* * * Dip. Sistemi e Informatica, Università di Firenze, Via Lombroso 6/17, 50134 Firenze, Italy,
E-mail A.Bondavalli@cnuce.cnr.it

Abstract

In designing high assurance systems, the dependability goals are achieved through the adoption of several fault tolerance techniques. Unfortunately, their combined effect on the system cannot be, in the general case, derived by straightforward composition of the stand-alone component's analysis, because of mutual dependence of their controlling parameters. In this paper the assessment of overall system dependability induced by such integrated fault tolerance organizations is carried out through a stochastic simulation approach. To this purpose, a few fault tolerant multiprocessor architectures, based on the integrated usage of standard error processing structures with a recently proposed diagnostic mechanism, called α-count, are selected and evaluated. The diagnostic mechanism gets its input (error signals) from the error processing mechanism, whose behaviour is in turn influenced by the rapidity and correctness with which α-count identifies permanently/intermittently faulty processors. The choice of the basic fault tolerance mechanisms to adopt, as well as the reference system architecture, has been driven by the characteristics of the envisaged target applications: mainly, stringent dependability requirements, to be traded with adequate levels of performance and cost. The analysis has been focused on performability, which is an appropriate measure to evaluate whether a certain design is "better" than another under dependability and performance point of view.

Keywords: High Assurance, Fault Tolerance, Multiprocessor Systems, Dependability, Performability Evaluation, Stochastic Simulation.

1 Introduction

A common approach to fulfil high assurance requirements consists in introducing redundancy into the system, managed by fault tolerance techniques, namely error processing and fault treatment [1]. Error processing aims at removing errors from a computational state, possibly preventing failure; fault treatment aims at preventing faults from being activated again, i.e. resulting in new errors. Dynamic error processing schemes (SCOP [2], stand-by sparing), which employ resources according to the actual fault manifestations, are particularly suited to cope efficiently with errors. On the fault treatment side, it has long been recognised that real systems experience mostly transient faults [3] which rapidly disappear; hence, removing a processor exhibiting a possibly transient faulty behaviour may unduly impair subsequent performance. In a recent paper [4] the same authors have presented a diagnostic mechanism based on a weighted count of the accumulated error scores of each processor, called α-count, designed to discriminate intermittent and permanent faults against low rate, low persistency transient faults. The mechanism, which draws from error signals available in the system, has been extensively analysed in terms of purposely defined performance figures; however, these figures are not directly related with classical dependability attributes.

Extensive studies on the development of a wide number of fault tolerance (FT) mechanisms have appeared in the literature, together with evaluation of the system benefits (in terms of various dependability attributes) deriving from the application of each single mechanism (e.g., [5-8]). However, the pre-set system dependability is generally achieved through the adoption of more than one FT technique; unfortunately their combined effect on the system cannot be, in the general case, derived by some sort of straightforward composition of the stand-alone component's analysis. In fact, the usual case is that the mechanisms contributing to fault tolerance have strict relationships among each others, to better exploit the synergy derived from their compound usage. The established relationships have, of course, to be properly accounted for when evaluating the overall system dependability figures; the resulting analysis is far more complex than if the mechanisms were independent from each other.

The contribution of this paper is mainly a comprehensive analysis of a few integrated fault tolerance organizations, to gain insights in the effects on the overall system dependability induced by such combined usage. Specifically, the α-count mechanism has been coupled with a few representatives of dynamic error processing schemes to build complete fault tolerance strategies.

The resulting fault tolerant organizations have been studied via simulation, to evaluate the performability gained by applying each individual strategy in the set. The envisaged target applications are characterized by high assurance requirements; however, they also call for adequate system organizations to trade the demand on high dependability figures with specific requirements on performance and cost. Therefore, the context of a multiprocessor architecture is assumed, since: i) owing to its natural redundancy, is particularly suitable both to provide large raw computational power and to implement techniques to tolerate operational faults; ii) its regular structure will best benefit of the increasing level of device integration at low cost. Both features fit well the requirements of evolving fault-tolerant embedded systems. Preliminary results of this study have been presented in [9]; here a comprehensive report is given, with ample presentation and discussion of results.

The rest of the paper is organized as follows. Section 2 describes the logical structure of the assumed multiprocessor architecture, identifying the components in charge of fault tolerance activities. Then, Section 3 introduces the fault tolerant organizations, recalling the behaviour of the α-count mechanism for fault discrimination and briefly describing the error processing structures selected to be coupled with α-count. The simulation environment set up for the evaluation of the overall fault tolerant architectures is presented in Section 4, including the fault model and the interesting quantities under analysis. In Section 5, a number of representative system scenarios are defined and analysed, followed by a thorough discussion of the obtained simulation results. Finally, conclusions are drawn in Section 6.

2 System Model

The architecture we assume is a symmetrical multiprocessor system (SMP) composed of N processors (or units), $u_0,..,u_{n-1}$, which are considered atomic failure units. The architecture also includes components devoted to manage dependability aspects, namely: i) proper allocation of tasks to processors and removal of faulty processors, ii) error processing and judgement on processors' behaviour, and iii) identification of faulty processors to be consequently removed. The logical architecture of our system is illustrated in Figure 1. Upon service request, the Planner

component selects a certain number of processors based on the error processing strategy[1] employed and makes available to them not just the input data but also the application software to run. Then, the task is executed by the selected processors. The correct execution of the same task with the same inputs on several processors, or by the same processor at different times always yields the same output. The results produced by the processors involved in the execution of the same task are collected by the Error Management (EM) component, which selects the result to be delivered by applying an adjudication function, and either forwards it to the users or stores it in a stable storage to be used in subsequent computations. When dynamic error processing mechanisms are employed [2, 10], redundant execution of an applicative task might be performed in phases, where the execution of further copies of the application is conditional on the absence of an adjudged result in the current phase, as notified by the EM; this implies information exchange between EM and the Planner. EM provides also information to another component, the Diagnosis Mechanism (DM): for each redundant task execution EM delivers to DM a notification about the processor(s) that originated disagreeing results with respect to the adjudicated output.

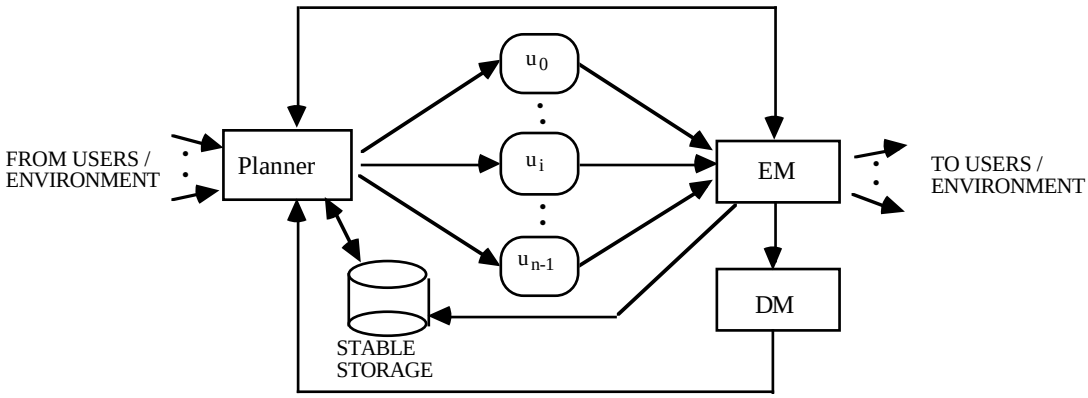

Figure 1. Logical architecture of the system.

Since most errors are caused by transient faults, DM is in charge to decide if the error frequency exhibited by a processor, and the attending loss of computing resources to recover the error's

[1] This number of processors selected, i.e. the amount of redundancy used, may vary for different applications if they prove to have different levels of criticality. However we will not consider this possibility in this work.

effects, are large enough to offset the performance benefits of keeping that processor on-line.[2] Upon taking such decision, DM tells the Planner that the processor must be removed from the system. The Planner then stops sending that processor any further service request. Repair/replacement of faulty processors is not considered in this paper; therefore, the number of active processors in the system progressively shrinks, and we assume that the system stops to be operative when the number of active processors decreases below a given threshold. Embedded systems that must operate for all their life in circumstances where no repair is admissible (for example, space-exploring vehicles) obviously fit the present system model. It also applies to a single mission of the much more popular mission-oriented systems, characterized by preset duration operative periods alternating with off-line periods, where preventive maintenance is regularly carried on (e.g., in the case of totally autonomous battle-field vehicles).

3 Fault Tolerant Organizations

In the assumed architecture, fault tolerance is managed essentially by the EM and DM components. Four different organizations, obtained by coupling in turn four different error processing schemes with the diagnostic mechanism α-count, have been chosen for evaluation in the following Sections.

3.1 The Diagnosis Mechanism

The DM component is in charge of identifying processors to be taken off-line. Physical faults are classified as permanent, intermittent or transient [1, 3]. Permanent and intermittent faults are internal, persistent faults, while transient faults are caused by external conditions (e.g. storm-originated EMI), whose effects tend to disappear rapidly. Since real systems experience mostly transient faults, removing a processor exhibiting a transient faulty behaviour may unnecessarily harm the system performance and diminish its residual redundancy. Therefore, processors showing erroneous behaviour have to be cautiously treated, because of the high percentage of transient faults with respect to the overall fault manifestations. The α-count mechanism, used as DM, is a simple count-and-threshold scheme presented in [4, 11], which was designed to dis-

[2] In general, DM would be fed by a multiplicity of error detectors. The present limitation tends to highlight the interplay between EM and α-count; conservative results will anyway be obtained.

criminate intermittent and permanent faults against low rate, low persistency transient faults. α-count adds up signals caused by errors in a given processor, as time goes on, weighing down signals as they get older. If the rate of detected errors, filtered according to a tuning parameter, exceeds a tunable threshold, the processor is signalled to the Planner.

The basic formulation of the filtering function α is the following:

$$\alpha_i^{(L)} = \begin{cases} \alpha_i^{(L-1)} \cdot K & \text{if} \quad J_i^{(L)} = 0 \\ \alpha_i^{(L-1)} + 1 & \text{if} \quad J_i^{(L)} = 1 \end{cases} \qquad 0 \le K \le 1 \qquad (1)$$

$$\alpha_i^{(0)} = 0$$

where α_i is a score associated to each not-yet-removed processor u_i to record information about the errors experienced by u_i and $J_i^{(L)}$ indicates the L-th error notification by EM on u_i: $J_i^{(L)} = 0$ in case of success, $J_i^{(L)} = 1$ upon error, and K is a parameter to be set by the system designer.

When the value of $\alpha_i^{(L)}$ exceeds the given pre-set threshold α_T, u_i is diagnosed as affected by a dangerously frequent intermittent fault, and the consequent signal is issued. The values that the parameters K and α_T have to be assigned so that the strategy works best, i.e., it recognizes faulty processors as soon as possible and lowers the probability of identifying healthy processors as faulty, depend on the expected frequency of permanent, intermittent and transient faults and on the probability of correct judgements of the error signalling mechanism used.

The behaviour of α-count has been analysed in [4, 9, 11] as a stand-alone mechanism. Two figures of merit have been evaluated: i) the time, D, between a (permanent or intermittent) fault occurrence in a processor u_i, and its recognition by the threshold crossing of the pertinent count α_i; ii) the wasted time, NU, spent by a processor idled after being wrongly signalled as faulty (normalized to the expected processor's life). Note that in the time span measured by D a faulty processor is maintained in use and relied upon, because its condition has not yet been recognized, thus opening a window of vulnerability to cathastrophic multiple fault occurrence.

In this paper we take advantage of the already performed analysis of α-count in choosing some parameter values in the ensuing evaluation.

3.2 The Error Manager

The EM component deals with processors' run-time errors. Some significant error-processing approaches are hereafter briefly recalled. We consider instances of dynamic error processing schemes which employ a relatively low level of redundancy; these solutions appear to be the most appropriate ones for our target high assurance applications, which require a good balance between dependability and performance (minimum usage of redundancy to favour performance, while assuring an acceptable level of dependability). The schemes are based on redundant execution followed by comparison (or, more generally, by adjudication, whenever hardware diversity is employed). As long as the occurring faults allow the adjudication of the correct result, errors generated by single processors are prevented to corrupt the output. This approach brings the advantage, in our context, of immediate identification of processors whose output disagrees with the adjudged one (i.e., processors to be notified to DM).

Self checking pair (SCP). In the self checking structure, two replicas of the task are executed by two different processors, possibly at the same time, and their results are compared by the adjudicator. Consistent results are accepted and a "correct" judgement is attributed to both processors. If a disagreement is observed instead, a detected error is notified as the service output. Then, a diagnostic routine is launched on each processor. If both diagnostics give the same output, whether "faulty" or "good", both processors are signalled as possibly faulty, since it is not possible to discern which one(s) actually failed. If instead only one processor recognizes itself as faulty, the other is not signalled to the DM. The fault coverage of the diagnostic routines plays an important role in the overall efficacy of DM: intuitively, the higher c_d, the higher the probability of correct identification of faulty processors, with a consequent more precise information transmitted to DM.

In case only one processor is identified as incorrect by the diagnostic routine, the computed output of the other "good" processor could optionally be assumed as the task result. A system configuration with this option enacted will be referred to as "Self-Checking Pair with Error Correction", or SCP-EC.

"2+1". This technique is a simple instance of SCOP (Self-Configuring Optimistic Programming) [2], a class of error processing strategies based on the idea of using redundancy incrementally over a number of steps. "2+1" employs three replicas; its execution starts with two replicas executed on two different processors and their results are accepted if they prove in agreement;

otherwise, the third replica is executed and its result examined by the adjudicator along with the other two. A correct result is then produced, provided that a single fault occurred, and the disagreeing processor is identified and signalled to DM. From its operational behaviour, "2+1" can also be seen as a "dynamic" variation of the traditional TMR ([3]) to improve efficiency.

“2+2”. The third scheme considered, called “2+2”, belongs to the same class as the "2+1" scheme. It is composed of four replicas, two of which are executed in a first phase. When two disagreeing results are observed, replicas are re-executed on two other processors, thus obtaining 4 results to build the judgement on. A result value is considered correct if agreed upon by a majority of at least two units.

“3+1”. In this scheme four replicas are also employed, with a different arrangement: three of them are concurrently executed in the first phase; if a majority of at least two agreeing results is not obtained, the second phase is entered, and all the four values concur in the choice of the final output, as in the “2+2”.

From the point of view of fault tolerance capability, “2+2” and “3+1”, employing more redundancy, obviously are the best, followed by “2+1” and by SCP. In fact, “2+2” and “3+1” are able to provide correct results not only in presence of 0 or 1 units failed (as “2+1”), but also in those cases where two units failed providing different results.

SCP exhibits fault tolerance only in its SCP-EC variant. This structure compares with the “2+1”, as the diagnostic routine executed upon mismatch is a 2nd phase of a sort, which offers added flexibility: its overall cost, both in terms of development and of execution time, can be modulated by design, in search of the best trade-off between fault coverage and probability of delivering erroneous results (the lower the catastrophic error probability, the higher the required coverage - and cost).

The basic SCP has fault detection capabilities only, with the consequent higher rate of detected errors as compared to the other schemes. It exhibits, however, the lowest probability of undetected error (excluding the “3+1”), limited to the probability of having two coincident faults causing identical results.

Regarding the level of redundancy used, SCP is the most convenient (provided that simple and short diagnostic routines are used). The cost of executing additional replicas is paid by the “2+1” and “2+2” schemes only in the (rare) event that a fault is actually detected (SCOP is in fact an

"optimistic" redundancy scheme). The “3+1” scheme pays always the cost of three executions in the first phase, in exchange of avoiding the need of the second phase for all single-error occurrences in the first one, which is instead entered by “2+1” and “2+2”. From the point of view of simplicity of operation, SCP is the favourite one, since it does not require to track and synchronize the outputs of replicas from phase to phase. Then, the simpler overall organisation of SCP has to be weighed against the lower offered tolerance. The “3+1” is the most expensive in terms of employed resources; however, it is characterized by a short time consumption. Moreover, it can deliver better fault-tolerance, provided it is equipped with a more sophisticated adjudicator (assuming that the adjudicator follows the three-out-of-three rule in the first phase, and the two-out-of-four in the second phase, two faults causing coincident errors are detected).

4 Evaluation

By coupling the fault detector mechanism α-count with one of the error processing schemes introduced in the previous Section, a complete fault tolerant organization is obtained. Its evaluation is performed hereafter. The intricate relationships existing among the integrated mechanisms EM and DM, that depend, among other things, on their respective parameters, make the analytical evaluation quite hard. In fact, note that: i) a Stochastic Activity Network (SAN) describing a single α-count mechanism expands to several thousands of states; ii) each N-ple of processors needs to be tracked individually in its behaviour, complete with the pertinent tuple of α-counts; iii) an N-ple assigned a given task may be composed of different processors in different executions (e.g., because of faulty processors being replaced); processor’s history thus interleaves with that of N-ples. An analytical model of such a system would count states in the number of millions or tens thereof; the necessary computational effort would have been beyond our resources. Therefore we have resorted to stochastic simulation, which is a valid tool for the class of redundant configurations excluding ultra-dependable systems. We developed our own discrete event, artificially regenerative simulator, which has been designed to deal with the multiprocessor architecture described in Section 2.

4.1 Fault Assumptions

The assumptions that define the fault model adopted in the evaluation are the following:

1- Only hardware faults are considered. Faults affecting the links of our architecture, the stable storage, the Planner, EM and DM components are not considered in this study.

2- All the processors in the system have the same constant fault rates λ_p, λ_i and λ_t for permanent, intermittent and transient fault respectively, and they fail independently by each other. The total fault rate is $\lambda = \lambda_p + \lambda_i + \lambda_t$. Permanent and intermittent faults last forever, i.e. no repair action is considered. Transient faults are momentary malfunctions due to external causes.

3- A permanent fault causes an error resulting into an incorrect service at every execution from its occurrence on. An intermittent fault, after its occurrence, is activated, i.e. results into an error, with a rate λ_m. Normally, $\lambda_t << \lambda_m$. A transient fault gives rise to an error lasting only for one task execution.

4- When two processors provide two erroneous results in executing the same task, their results will have an identical erroneous value with a probability q_d.

5- The diagnostic routines used by the SCP organizations never identify a healthy processor as faulty; therefore the coverage parameter c_d only refers to the likelihood of the diagnostics to correctly recognize a faulty processor as faulty. Actually, this assumption is verified by diagnostic routines many current systems are equipped with.

6- The outcomes of a redundant execution may be: i) *success*, i.e., the delivery of a correct result, ii) a *detected error*, detected either by comparison of redundant results or by a (reliable) watchdog timer catching timing errors, or iii) an *undetected* (or catastrophic) *error* (delivery of an erroneous result).

4.2 Performability Measure

Our work is directed to [real world] high assurance systems supporting critical applications for which reliability or safety alone are not the only or topmost design concern; performance (in terms of the number of error-free task executions in a given time/mission) and overall system cost are also ruling parameters. For such systems, a performability measure [6, 12, 13] is more appropriate to evaluate whether a certain design is “better” than another. Necessary to performability is the definition of a reward model, which ideally should take into account all factors intervening in the production of valuable results, or in operational costs. To limit the

complexity of the analysis we use here, by way of example, two simple additive reward models which fit our mission-oriented systems, but do not capture all the utility factors of the computational schemes, e.g., the organizational simplicity of the SCP and the promptness of the "3+1".

In the first model, M_1, successful executions add one unit to the value of the mission; executions producing detected errors add a cost C_B; an undetected error aborts the mission (mission failure), and sets the reward to a negative value C_C. This reward model is appropriate, for instance, in tool control in manufacturing industry, where each iteration produces a unit of some product, the loss associated to an undetected error may be the stoppage and/or some damage to the tool, and that of detected error may be the production of an imperfect item. An alternative applicative scenario is a somewhat complex transaction-processing or scientific application, where detected errors can be recovered (at some cost); an undetected error destroys the value produced during the day, also implying additional costs (e.g., costs from erroneous services provided to clients, or from a roll-back and rerun of the transactions at the end of the day, after some inconsistency has been detected by external means).

$$M_1 = \begin{cases} \text{"no. of successes"} - \text{"no. of detected errors"} \bullet C_B, & \text{if mission success} \\ -C_C, & \text{if mission failure} \end{cases}$$

In the second model, M_2, both detected and undetected errors cause the failure of the mission, and a cost (possibly high) is associated to this event. With respect to the previous performability structure, M_2 fits more reliability-oriented applications, where there is no possibility to recover any error, even the detected ones.

$$M_2 = \begin{cases} \text{"no. of successes"}, & \text{if mission success,} \\ -C_C, & \text{if mission failure.} \end{cases}$$

4.3 Simulation settings

System characteristics adopted in the simulation experiments are here described. The system load has been assumed to be infinite, and the processors are used without any synchronisation or scheduling delays, to reach the highest parallelism permitted by the employed error-processing scheme. Table 1 summarizes the notation and values used for the parameters involved in the simulation.

Parameters description	**Symbols**	**Values**
Number of processors in the system	N	10
Minimum number of processors for the system to be operational	N_{min}	6
Processors fault rate (per hour)	λ	5E-04
Permanent fault rate (per hour)	λ_p	$0.05\times\lambda$, $0.03\times\lambda$
Intermittent fault rate (per hour)	λ_i	$0.15\times\lambda$, $0.07\times\lambda$
Transient fault rate (per hour)	λ_t	$0.8\times\lambda$, $0.9\times\lambda$
Task duration (sec)	T_t	600
Probability of coincident error given that two processors failed executing a task	q_d	0.04
Activation rate (per hour) of intermittent fault	λ_m	1.8
Fault coverage of diagnostic routines	c_d	0, ..., 0.99
Cost of a detected error	C_B	variable
Cost of mission failure	C_C	variable
Threshold for considering a processor faulty	α_T	0.5,1.1, 2, 3,INFIN
Ratio by which α_i is decreased after a success	K	0.995
Mission length (hours)	T_M	1500, 2500

Table 1. Parameters and values used in the simulation

Processors fault rate λ has been chosen to be 5×10^{-4} per hour; then, two different settings for λ_p, λ_i and λ_t have been analysed, to investigate the sensitivity of the scheme to transient faults occurrences. Our multiprocessor system is supposed to run tasks having the same duration, T_t, which has been set to 600 sec. From the analysis of the α-count mechanism's behaviour performed in [4, 11], where the relation between the activation rate of intermittent faults and the decay ratio of the filtering function (K) has been studied, a significant pair of values has been chosen for the couple (λ_m, K). Missions of pre-set time duration (T_M) have been considered, having as termination conditions: i) a the delivery of an undetected error; ii) the depletion of processors (less than N_{min} left); iii) the expiration of T_M. The main parameters which have been left variable are a) the α-count threshold α_T, b) the ratio of transient to intermittent/permanent faults occurrence, and c) the fault coverage and execution time (c_d,t_d respectively) of the diagnostic routines used in the SCP scheme. Of course, the duration of a diagnostic routine increases with increasing coverage, and the chosen law which keeps the two parameters tied is the following: for $c_d\leq0.6$, t_d =180 sec; for $0.7\leq c_d<0.9$, t_d =600 sec; and for $c_d\geq0.9$, t_d =3600 sec. To fully appreciate the influence of the α-count mechanism, proper values to α_T have also been used to

resemble system configurations where this mechanism is not present. In these conditions, the following two policies have been implemented towards the treatment of faults: components are removed at their first erroneous behaviour regardless of the fault nature (α_T = 0.5) and faulty components are never removed (α_T = INFIN, where INFIN is a suitably large value which depends on how many tasks executions are allowed by the mission duration). The simulation results have been obtained with 90% confidence level within interval relative half-width of at most 10%.

The values just shown for the simulator settings are not intended to refer to any particular system; rather, some parameters, e.g. the fault rates, are representative of average values exhibited by a broad range of real systems; while others, as the mission times and confidence level, have been chosen to trade off precision and significance of simulated results vs. computational resources.

5 Simulation results

The results of the simulation experiments are now presented and discussed. In order to gain deeper understanding of the mutual effects of the error processing and the α-count mechanisms composing each selected fault tolerant organizations, three additional quantities other than the performability measure, which is the main focus of this work, have been evaluated. Indicating with FU a processor affected by a permanent/intermittent fault, we have evaluated: i) the average number of processors identified as FUs; ii) the average number of processors among those removed which are really FUs; and iii) the average number of healthy processors wrongly identified as FU, and so removed. Two scenarios have been studied, which differ in: a) the mission duration, b) the conditions which determine the mission failure, c) the reward structure and d) the ratio of transient to intermittent/permanent faults occurrence.

The coverage c_d of the diagnostic routine in the SCP scheme gives a further dimension to the analysis to be carried out, which is not present in the other schemes; examining extensively all the value combinations for the scheme's parameters could hamper the visibility of relevant results. Therefore we first analyzed the c_d parameter's effects on the SCP, in the intent of simplifying the overall study.

5.1 Study of the Self-Checking Pair in Terms of c_d

The effects of the diagnostic coverage c_d have been analysed in terms of a few significant figures of the system behaviour, namely the probability of mission success, the average number of healthy processors unduly removed and the performability. Figure 2 shows the results of this analysis. In the figure, c_d =0 accounts for the extreme case where a faulty processor would never be recognized as such, as in the bare SCP (i.e. without error diagnostic routines). Values of c_d lower than 0.5 have not been considered, as diagnostic tests having coverage in this range are of no practical interest.

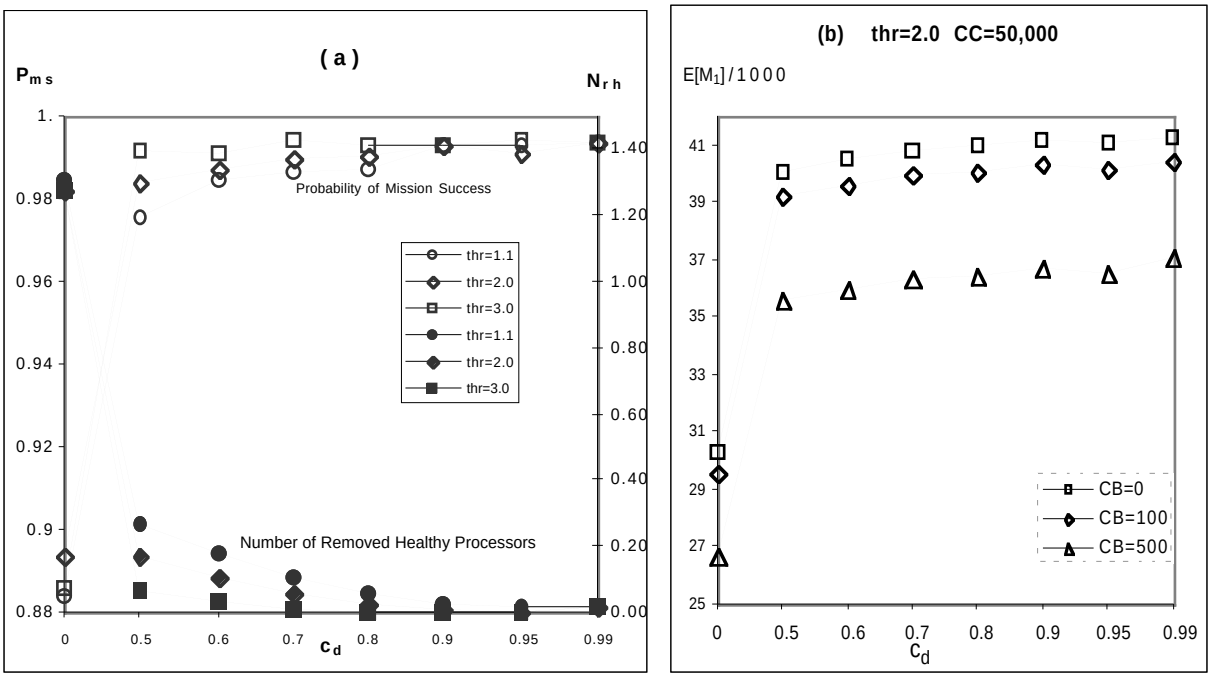

Figure 2. Effects of the Diagnostic Coverage Parameter c_d

In more details, Figure 2(a) plots the probability of mission success (P_{ms}, on the left ordinate axis) and the average number of removed healthy processor (N_{rh}, on the right ordinate axis), at values of c_d in the range [0,0.99], for several values of the threshold α_T. In Figure 2(a) it can be seen that both P_{ms} and N_{rh} markedly improve from c_d=0 to c_d=0.5; then their values get limited increase, with no valuable improvement with c_d above about 0.9. P_{ms} and N_{rh} get better as the parameter α_T increases, in the chosen range [1.1, 3.0]; again, for c_d > 0.9 scant gain is obtained.

Figure 2(b) plots the performability based on the reward structure M_1 for three values of the cost, C_B,of the benign error, a value of 50,000 for the cost of catastrophic error, and a threshold α_T=2.0. As in the part (a), the presence of diagnostic tests brings a huge improvement in

performability, even at low coverage. The curves show then a slow variation with c_d above 0.7-0.8. Of course, the influence of the cost C_B is quite visible, while not altering the curves' shape.

The SCP-EC variant improves on the SCP in delivering a lower number of detected errors: when only one processor is tagged as faulty out of the two diagnostics run on the two processors, the result provided by the "good" one is delivered as a correct output. The down side of this option is that it comes with higher probability of undetected error: because of the incomplete diagnostic coverage, the possibility arises that in case of both processors failing with non-identical results, one of the twin processors does not recognizes its own fault. In such a case, rare but to be accounted for, its result is wrongly assumed correct. This suggests to use SCP-EC with an adequately high value of c_d.

To get an idea of the impact of the peculiarities of the SCP-EC w.r.t. SCP, we made an evaluation of the performability E[M1] yielded by the two organizations, under the setting c_d=0.9, α_T=2.0, C_C=50,000, for three values of C_B.

	C_B=0	C_B=100	C_B=500
SCP	41195	40288	36664
SCP-EC	41102	41010	40646

Table 2. Performability E[M1] of SCP and SCP-EC, with c_d=0.9, α_T=2.0, C_C=50,000

The obtained results are reported in Table 2. It can be noted that SCP shows slightly better values for C_B=0; however, the lower number of detected errors experienced by SCP-EC brings noticeable advantages to SCP-EC at increasing values of the cost C_B.

The above discussed results suggest that comparisons with the other fault-tolerant organizations can be carried out considering a fixed value for c_d. Since a coverage of 0.8 can be attained by relatively simple test programs, this value has been chosen for SCP in the following full range of simulations; this way, one of the merits of the SCP structure, that is its simplicity and low cost, has been preserved. This feature has to be weighed in the final assessment of the current analysis' results, whenever the merit figures of the SCP structure are comparable with those of competing solutions. Comparisons with the SCP-EC variant have been also performed, limited to the setting $c_d = 0.9$ and $\alpha_T = 2.0$.

5.2 Scenario I: short missions and detected errors do not cause mission failure

In this scenario, T_M=1500 hours and missions terminate with failure whenever an undetected error occurs or when less than N_{min} processors are left. The two fault tolerant organizations composed of SCP plus α-count and "2+1" plus α-count have been considered. Transient faults account for 80% of the total faults (λ_t=0.8×λ), intermittent fault rate is λ_i=0.15×λ, and permanent fault rate is λ_p=0.05×λ.

Figure 3 shows the performability values obtained adopting the reward structure M_1, at varying values of the cost C_B of a detected error.

In Figure 3.a, the value assigned to C_C is zero, to stress the impact of the higher number of detected errors raised by SCP w.r.t. "2+1". The curves relative to SCP are drawn against the left and bottom axes, while those relative to "2+1" are referred to the right and top axes; in the latter case, curves and axes are drawn as dashed lines for better clarity. The bottom abscissa covers values of C_B from 0 to 450; at this point, the performability of SCP drops about 10% out of the value obtained with C_B=0. As the dashed curves show, in the upper part of the figure drawn in logarithmic scale, a similar 10% performability drop occurs for the "2+1" at C_B values ranging in the millions. The superiority of the "2+1" over the SCP stems from the very low number of detected errors in successful missions exhibited by the "2+1". However, in applications where benign errors have a relatively low associated cost, the SCP structure may be usefully adopted, because of its structural simplicity.

For growing values of C_B, the effect of the higher number of detected errors experienced by SCP w.r.t "2+1" shows up: since the average number of detected errors for SCP are a few thousands times that of the "2+1", the performability obtained by the SCP sharply drops whilst the "2+1" curve hovers straight above. The behaviour of the SCP-EC configuration departs significantly from that of the simple SCP. Not unexpectedly, from the figures obtained for that scheme it appears more similar to the "2+1" than to the SCP. As already discussed in the previous section, the most prominent variation against the SCP is in the number of detected errors, which gets reduced by a factor of ten, all other parameters being the same. In our setting, this strong reduction brings notable advantages even for low values of C_B.

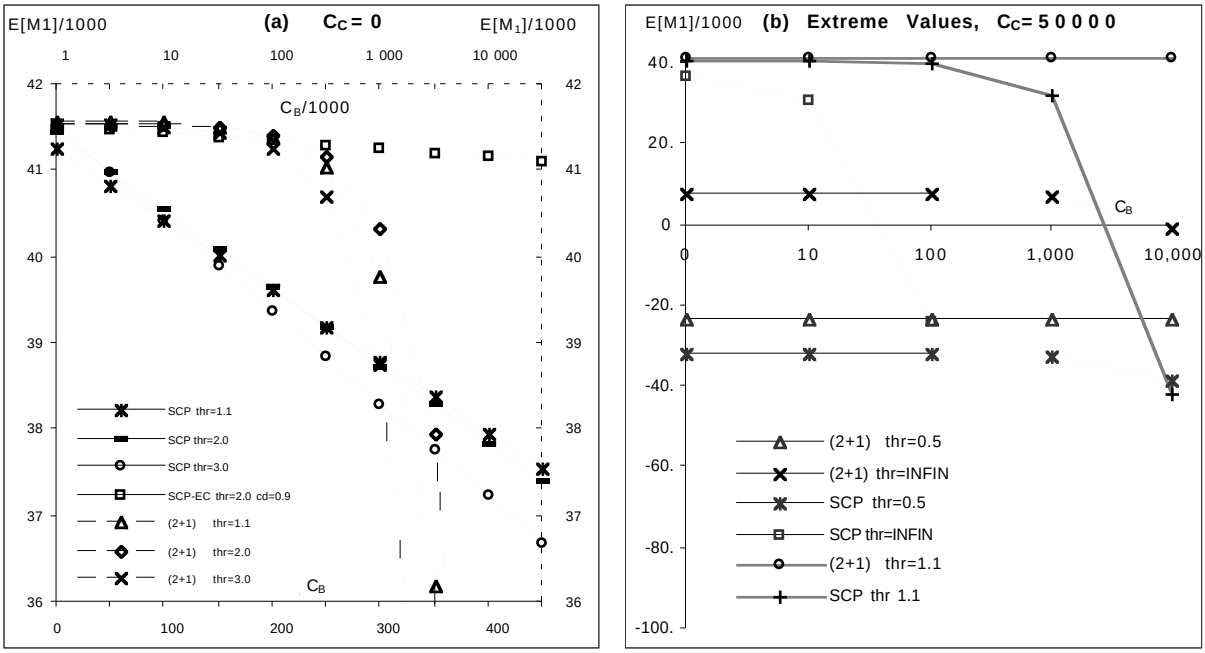

Figure 3. Performability vs. the cost C_B of detected errors

In the curves relative to SCP and"2+1" threshold values α_T =1.1, α_T =2.0 and α_T =3.0 have been considered, which constitute the interesting range to appreciate the effect of the usage of the combined error processing and α-count mechanisms. In the SCP case, the sensitivity to variations of α_T is appreciable in the whole displayed range, especially for higher costs of detected errors. Notably, the effect of performance inversion that clearly shows up is caused by the typical α-count behaviour: the curve with the larger value of α_T starts higher, only to get worse performability at increasing C_B costs, because faulty processors are kept longer in the system, raising the chance of detected errors. The "2+1" is not sensible to the variation of α_T in the range [1.1, 3], originating almost overlapping plots, for values of C_B extending through ~100,000. At larger values, the expected dependence on α_T is visible; the best results are obtained with the intermediate value α_T =2.0.

To examine the effect of the cost C_C on performability, recall that such cost is incurred whenever the mission fails. Although not shown in the figures, the simulation results show that SCP and "2+1" compare well in reliability, as they achieve similar probability of mission success. The best figures are practically the same in both schemes, albeit obtained with different parameters: the "2+1" scores 0.9944 with α_T =1.1, the SCP gets 0.993 with α_T=3.0. Therefore, the effect of C_C on performability is comparable for both schemes, at least in a wide range of values (in our case, up to ~100,000). Of course, as higher values of C_C would have to be paid, even the small difference in the success probability would eventually become sensible.

In Figure 3.b the extreme cases are shown, emulating the system behaviours in absence of α-count. With α_T =0.5, i.e. a value smaller than the elementary α-count increment of 1, even a single, solitary error gets the count over the threshold, causing the processor removal. Conversely, an infinite value for α_T means that processors are never removed, no matter how many times they incur in errors. Figure 3.b has been obtained by using C_C=50,000, to give a more complete view of the extreme situations (reasonably, a catastrophic error has a weight greater than that of a detected error). The first observation is that α_T=0.5 always yields negative performability in both schemes, with worse values in case of SCP. The unforgiving removal of processors, even upon a transient fault, causes most missions to terminate unsuccessfully because of depletion of processors (and SCP, because of the assumed value of diagnostic coverage, is more prone to point out processors as faulty). In the other extreme case, labelled α_T=INFIN in the figure, SCP gets better performability at low values of C_B. In fact, as in this case faulty processors are never removed, their continued presence in the system is less harmful for SCP then for "2+1" (this last has more chance to encounter two faults during a task execution than the former because of the two phases, thus resulting in a higher probability of undetected error). However, although the number of missions completed by SCP is higher than that of "2+1", the number of detected errors in completed missions is much higher for SCP than for "2+1". Therefore, for growing values of C_B, the performability of SCP heavily degrades, getting negative values. Curves with a minimal level of the threshold α_T , i.e. 1.1, have been drawn both for SCP and "2+1", to give visual and quantitative evidence of the benefits carried in by α-count, even in its "lightest" implementation.

		α_T=1.1	α_T=2	α_T=3
SCP	faulty processors	1.380	1.382	1.398
	removed faulty processors	1.379	1.381	1.395
	removed healthy processors	0.0534	0.0238	0.0030
2+1	faulty processors	1.3898	1.4038	1.3796
	removed faulty processors	1.3880	1.4026	1.3766
	removed healthy processors	0.0114	0	0

Table 3. Simulation results w.r.t. the identification of faulty processors

Table 3 summarizes the results w.r.t. the removal of processors. In both organizations practically all faulty processors are identified and removed, and the number of healthy processors wrongly

removed is negligible, even with low values of the threshold α_T. The number of faulty processors changes with α_T because of the different mission duration caused by α_T .

To explore the sensitivity of the selected fault tolerance strategies to variations of the percentage of transient faults, another set of simulation runs have been carried out, keeping the previous value of the total fault rate λ, while changing the ratio of transient faults to intermittent/permanent faults. Chosen values for λ_p, λ_i and λ_t, have been $0.03\times\lambda$, $0.07\times\lambda$ and $0.9\times\lambda$ respectively. Figure 4 has been plotted following the same criteria adopted in Figure 3 to allow direct comparisons.

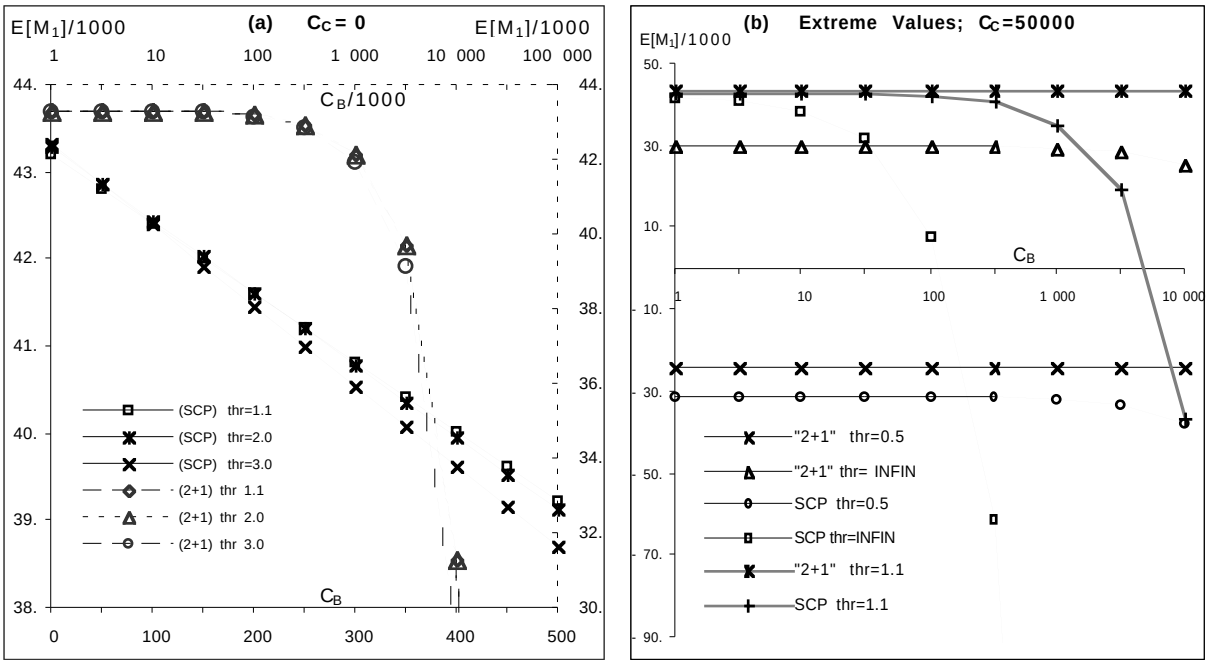

Figure 4: Performability as a function of the cost C_B of detected errors

Looking at Figure 4 it can be immediately observed, as expected, an improvement shown by both strategies. In fact, the lower rate of permanent and intermittent faults increases the probability of successful missions. As shown in Figure 4.a, the trend of the curves remains essentially the same. In fact, as derived from the analysis of α-count, the ability of this mechanism to correctly discriminate faults highly depends on the difference between the rate of transient faults and the manifestation rate of intermittent faults: the higher is this difference and the better is the discrimination performed by α-count. The value chosen for λ_t in the present case does not significantly affect such difference w.r.t. the first case. In Figure 4.b, while there is no appreciable difference with Figure 3.b for α_T=0.5 (as expected, since the total failure rate has not been changed), it can be observed a marked improvement in "2+1" with α_T=INFIN: the reduced

presence of permanent/intermittent faults significantly diminishes the probability of executing the second phase for this strategy.

Table 4 shows the behaviour of SCP and “2+1” with respect to the identification of faulty processors in the present case.

Numbers in Table 4 are smaller than the corresponding ones in Table 3 because of the smaller number of faulty processors now present in the system. However, the same comment as in Table 3 applies here.

		α_T=1.1	α_T=2	α_T=3
SCP	faulty processors	0.7166	0.722	0.7398
	removed faulty processors	0.7148	0.7196	0.7386
	removed healthy processors	0.045	0.0142	0.0026
2+1	faulty processors	0.7222	0.7302	0.7356
	removed faulty processors	0.722	0.7298	0.7346
	removed healthy processors	0.0162	0.	0.

Table 4. Simulation results w.r.t. the identification of faulty processors - second run

5.3 Scenario II: longer missions, low ratio of transient faults and detected errors do cause mission failure

Scenario II differs from Scenario I in having longer missions, T_M=2500 hours, and also in including detected errors as cause of mission termination with failure. The longer mission duration and, especially, the catastrophic consequences of even detected errors suggest not to use SCP as error processing scheme. Hence, the fault tolerant organizations consisting of “2+1”, “2+2” and “3+1” coupled with α-count have been studied in this scenario. The fault rates are $\lambda_p = 0.05\times\lambda$, $\lambda_i = 0.15\times\lambda$ and $\lambda_t = 0.8\times\lambda$.

Two sub-scenarios have been individually studied. In the sub-scenario II.1 the fault tolerant organizations based on “2+1” and “2+2” have been considered: the goal is to see the effect on the performability induced by the higher fault tolerance of the "2+2". The sub-scenario II.2 has the goal to compare two schemes, i.e. “2+2" and"3+1", having the same fault-tolerance but different organization in resource usage.

With reference to sub-scenario II.1, Figure 5 illustrates the performability, based on the reward structure M_2, at varying values of the cost of a mission failure $C_{C,}$ for $\alpha_T = 1.1$ and $\alpha_T = 3.0$. The behaviour in the extreme cases (as defined in Section 4.3) is illustrated by four curves, plotted for α_T = INFIN and $\alpha_T = 0.5$ (the two curves with $\alpha_T = 0.5$ overlap). The best results are obtained by "2+2" with α_T=1.1; while this is not surprising, since "2+2" has an higher probability of successful missions than "2+1", it has to be noted that "2+2" is not always better than "2+1" in terms of performability: in fact, the curve describing "2+2" with α_T=3 is the lowest in the set.

The worsening of the performability at increasing values of α_T shown by both strategies makes explicit the conflicting effects of α_T on a) the (wrong) removal of healthy processors and b) the (risky) longer permanence of faulty processors in the system. In the general case, as discussed in [4, 11], increasing values of α_T decrease the number of wrongly removed healthy processors (as shown by Table 5), while the probability of erroneous computation tends to increase because faulty processors stay longer on-line. In the present setting, where wrong outputs (even if detected) cause costly mission failures, the second effect dominates. Moreover, since faulty units spend more time in the system as the threshold increases, the second phase, where "2+2" uses two processors instead of one, is entered more often: this contributes to the slightly better results of "2+1" with α_T=3.

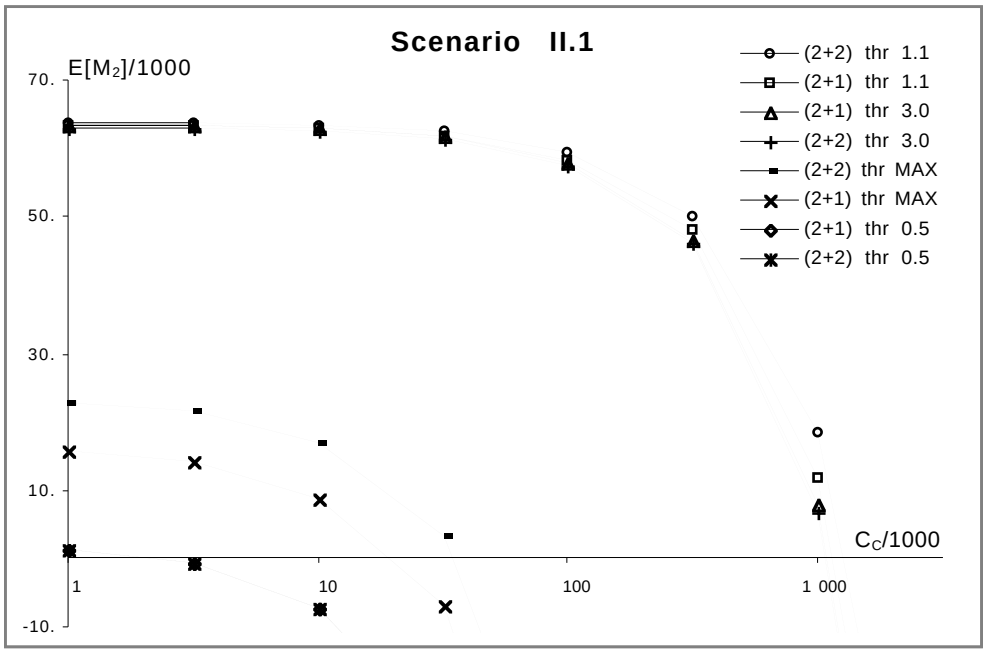

Figure 5. Performability in Scenario II.1, as a function of the cost C_C of mission failure

The curves relative to α_T =0.5 and α_T =INFIN give visual evidence of the strong α-count's effect on the performability, as it has been done in Scenario I.

Table 5 presents the results of sub-scenario II.1 with regard to the removal of processors. Both schemes achieve very good diagnostic precision, with negligible differences between them.

		α_T=1.1	α_T=3
2+1	removed processors	2.2034	2.1808
	removed faulty processors	2.1882	2.1808
	removed healthy processors	0.0212	0.
2+2	removed processors	2.2004	2.2018
	removed faulty processors	2.179	2.2018
	removed healthy processors	0.0214	0.

Table 5. Simulation results w.r.t. the identification of faulty processors in Scenario II.1

With reference to sub-scenario II.2, Figure 6 illustrates the performability, based on the reward structure M_2, in terms of the mission failure C_C, for α_T = 1.1 and α_T = 3.0, as well as for the extreme cases α_T = 0.5 and α_T = INFIN.

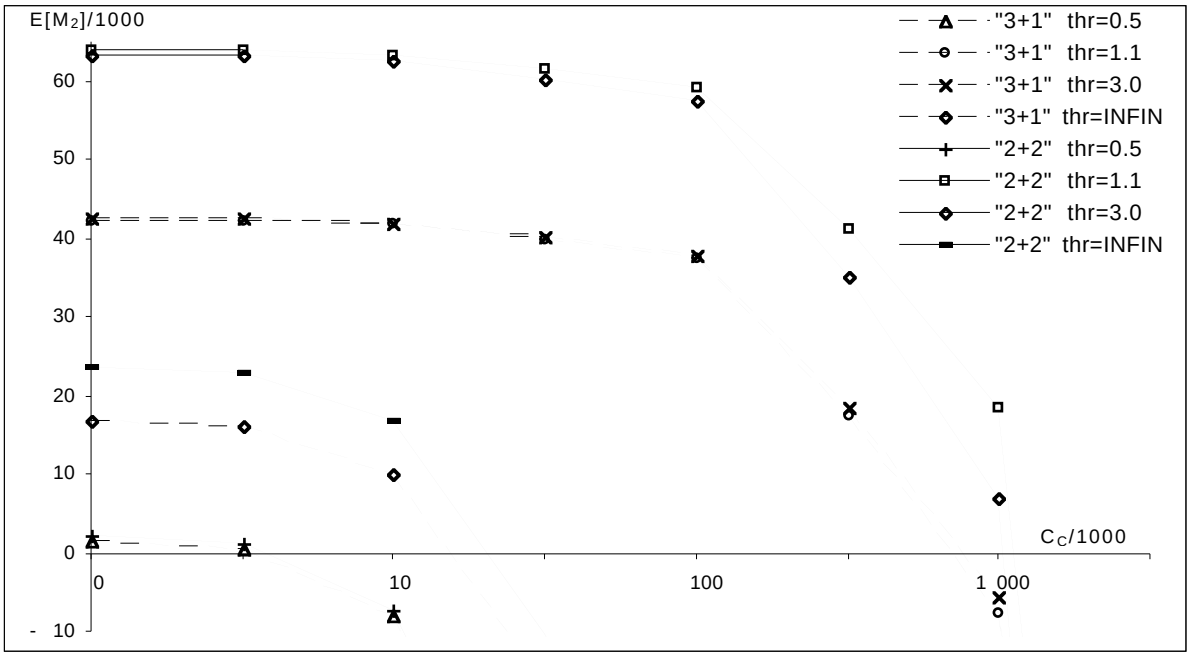

Figure 5. Performability in Scenario II.2, as a function of the cost C_C of mission failure

It is apparent from Figure 6 that, for low values of the failure cost C_C, the "3+1" delivers roughly one-third less performability than the "2+2". This is a consequence of the fact that the "3+1" is handicapped by having always to run at least three replicas, and, on the other hand, the reward model M_2 does not take into account the responsiveness, which is better in the "3+1".

Focusing on the effect of α_T, the figure shows that in the "3+1" the performability grows with α_T (in the value range examined), while in the "2+2" a degradation is observed when α_T =3.0 (as already pointed out when discussing scenario II.1). In fact, in this case, the longer permanence of faulty processors leads to a higher probability of mission failure for the "2+2" wrt the "3+1" because of the higher probability of the former to enter the second phase, during which the scheme is further exposed to the occurrence of a second fault. The figure shows that, if the cost of the failure is high enough, the performability of the "2+2" gets even lower than that of the "3+1"; however, in our settings this observation leads to no practical use, since this happens in a region where the performability is negative in both structures.

Table 6 completes the analysis of the fault tolerant organizations based on "2+2" and "3+1" w.r.t. the removal of processors. Because both strategies have the same fault tolerance capabilities, and consequently the same level of correctness of processors' judgements issued to DM, the figures pertaining the removed faulty processors shown by both strategies are almost the same (and, as expected, very good ones).

		α_T=1.1	α_T=3
3+1	removed processors	2.1962	2.1698
	removed FU	2.1756	2.1698
	removed healthy processors	0.0207	0.
2+2	removed processors	2.2004	2.2018
	removed FU	2.1790	2.2018
	removed healthy processors	0.0214	0.

Table 6. Simulation results w.r.t. the identification of faulty processors in Scenario II.2

6 Conclusions

This paper has contributed to the understanding of the effects of integrating fault tolerance mechanisms in a multiprocessor system, targeting high assurance applications, where

dependability and performance are of great concern, thereby focusing on the performability measure.

Starting from known error processing and diagnostic mechanisms, a few fault tolerant multiprocessor organizations have been examined. The computational power offered by multiprocessors, and the recognition that the great majority of physical faults affecting processors has a transient nature, led us to couple simple instances of error processing schemes based on redundant execution with the threshold-based fault detector mechanism α-count.

The analysis performed has shown how the behaviour of the combined use of error processing and α-count mechanisms (well analysed in isolation in previous work), influence a measure which is descriptive of the entire system, namely the performability. A simulation approach has been adopted, to overcome the difficulties stemming from the inter-dependencies of the selected fault tolerance mechanisms, which would result into a state number explosion in attempting analytical solutions. The early presentation made in [9] has been brought to full extent, with: i) ample reports and discussions of simulation data, ii) the inclusion of diagnostic routines in the SCP scheme (with detailed analysis of the effects of their coverage factor), iii) the addition of the "3+1" scheme, and iv) the analysis of sensitivity to the percentage of transient faults (in Scenario I).

The results of our simulations have given numerical evidence to qualitative behavioural forecasts. The conflicting effects of the threshold α_T on the (wrong) removal of healthy processors and on the risky longer permanence of faulty processors in the system, combined with the different fault signalling capabilities of the error processing schemes, have been clearly visualized. The utility of equipping the system with the α-count mechanism has also been shown up, by a direct comparison with extreme values of α_T to simulate simple fault treatment policies alternative to α-count (processor removal at the first error detection, and processors never removed). Finally, an important outcome of our analysis is that careful integration of the fault tolerance mechanisms has to be planned, as less-than-obvious results may be otherwise obtained. We have shown, in fact, that the intuitively "golden" choice of pairing α-count with the best error processing scheme ("2+2") actually performs worse than using the cheaper "2+1".

References

[1] J. C. Laprie, "Dependability - its Attributes, Impairments and Means," in *Predictably Dependable Computing Systems*, B. Randell, J. C. Laprie, H. Kopetz, and B. Littlewood, Eds.: Springer-Verlag, 1995, pp. 1-28.

[2] J. Xu, A. Bondavalli, and F. Di Giandomenico, "Dynamic Adjustment of Dependability and Efficiency in Fault-Tolerant Software," in *Predictably Dependable Computing Systems*, B. Randell, J. C. Laprie, H. Kopetz, and B. Littlewood, Eds.: Springer-Verlag, 1995, pp. 155-172.

[3] D. P. Siewiorek and R. S. Swarz, *Reliable Computer System - Design and Evaluation*: Digital Press, 1992.

[4] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni, "Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults," *IEEE Transactions on Computers*, vol. 49, pp. 230-245, 2000.

[5] J. Arlat, K. Kanoun, and J. C. Laprie, "Dependability modelling and evaluation of software fault-tolerant systems," *IEEE Transactions on Computers*, vol. C-39, pp. 504-512, 1990.

[6] S. Chiaradonna, A. Bondavalli, and L. Strigini, "On Performability Modeling and Evaluation of Software Fault Tolerance Structures," presented at EDCC1, Berlin, Germany, pp. 97-114, 1994.

[7] J. B. Dugan and M. R. Lyu, "Dependability Modeling for Fault-Tolerant Software and Systems," in *Software Fault Tolerance, Trends in Software*, M. R. Lyu, Ed.: John Wiley & Sons, 1995, pp. 109-136.

[8] Special Issue on Fault Tolerant Software, *IEEE Transactions on Reliability*, vol. 42, 1993.

[9] F. Di Giandomenico, S. Chiaradonna, A. Bondavalli, and F. Grandoni, "Evaluation of Integrated Error Processing and Fault Diagnosis in Multiprocessor Systems," presented at PDPTA '2000, Las Vegas, Nevada, USA, pp. 1145-1151, 2000.

[10] B. Randell, "System Structure for Software Fault Tolerance," *IEEE Transactions on Software Engineering*, vol. SE-1, pp. 220-232, 1975.

[11] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni, "Discriminating Fault Rate and Persistency to Improve Fault Treatment," presented at 27th IEEE FTCS - International Symposium on Fault-Tolerant Computing, Seattle, USA, pp. pp. 354-362, 1997.

[12] J. F. Meyer, "On Evaluating the Performability of Degradable Computing Systems," *IEEE Transactions on Computers*, vol. C-29, pp. 720-731, 1980.

[13] A. T. Tai, A. Avizienis, and J. F. Meyer, “Performability Enhancement of Fault-Tolerant Software,” *IEEE Transactions on Reliability, Sp. Issue on Fault Tolerant Software*, vol. R-42, pp. 227-237, 1993.