This paper has appeared in the proceedings of 3rd IEEE High Assurance System Engineering Symposium.

A new Heuristic to Discriminate between Transient and Intermittent Faults

F. Grandoni[1], S. Chiaradonna[2] and A. Bondavalli [2]

1) IEI-CNR, Via S. Maria, 46, 56126 Pisa, Italy, grandoni@iei.pi.cnr.it
2) CNUCE-CNR, Via S. Maria, 36, 56126 Pisa, Italy, a.bondavalli@cnuce.cnr.it, S.Chiaradonna@guest.cnuce.cnr.it

Abstract

Effective discrimination between transient and permanent faults is a very important practical problem in (dependable) system design. A count-and-threshold mechanism named α-count, designed to discriminate between transient faults and intermittent faults in computing systems, is presented in an enhanced embodiment. It retains enough simplicity to allow exhaustive analysability through simple models. It is shown that the introduction of two operating thresholds, instead of the single one present in the basic scheme already known, both improves the performance figures of the mechanism and eases the designer's task of tuning the internal parameters.

1 Introduction

Most faults occurring in computing systems are temporary in nature; more precisely, temporary physical faults can be distinguished into internal faults (usually termed intermittent) and external faults (known as transient). The former, caused by some internal part going out its specified behaviour, usually exhibit a relatively high occurrence rate after their first appearance, and tend to become permanent. Conversely, the cause of transient faults cannot be traced to a defect in a well identifiable part of the system. Discarding the affected component would be appropriate for intermittent faults, but may prove too radical a measure, provided the state of the computation can be someway saved, to be resumed later on (possibly by outright restart, as in most non-critical applications). In practice, coping with transient faults requires to find a convenient trade-off between: i) system throughput - keep ailing component on the workplace until death, and ii) system correctness - put offline suspect component to prevent unrecoverable errors.

In fact, a wide range of techniques, spanning from various retry schemes to rather sophisticated off-line error log audit and trend analysis have been used, or studied in the literature, to defer ousting components experiencing transient faults, with reference to specific systems. An idea which has long been exploited has been to count the number of fault events: the number getting "too large" in a given time frame would signal the system component at hand as ready to be removed. In a previous paper [2], a simple mechanism, named α-count, has been presented, with the aim to provide the designer a flexible tool to cope with the transient faults problem. α-count has a mathematically defined structure, light enough to result into simple models, easily analysable by automatic tools, yet providing wide applicability via parameters fitted out to tune its behaviour to system figures, such as CPU or input devices fault rates.

In the original α-count, a single decision was envisioned about a component, namely: keep it or trash it; therefore, such a decision directly influenced both sides of the trade-off described above. This effect may be attenuated if we could someway de-couple the need of not relying on the activity of a possibly faulty component, from its actual dismission. In this paper new α-count scheme has been devised for this purpose, where a component u is kept in full service as long as its score α is lower than a

first threshold α_L. When the value of α grows bigger than α_L, *u* enters a restricted operation mode, where it continues to run application programs, but its results are not delivered to the user or used by the system, other than to continue to feed α-count. If α continues to grow, it eventually crosses a second threshold α_H, with $\alpha_H > \alpha_L$: then the component *u is* considered affected by a permanent fault. If, instead, α, after some time, becomes smaller than α_L, the component is brought back in full service.

This scheme allows keeping the lower threshold at a low value, thus reducing the diagnosis delay. Healthy components being hit by occasional fault bursts are pushed into a non-utilisation period (i.e. until their α-count goes back under the lower threshold), but the probability of irrevocable ousting can be kept as low as desired by means of an appropriately high value for α_H.

2 Related work

Several techniques have been proposed to discriminate transient faults in computing (sub) systems. They can be broadly classified in two groups: (A) techniques designed to support human intervention, and (B) algorithm-based, automatic mechanisms.

(A) The solutions in this group mainly entail analysis of bulky error logs, by means of (possibly) sophisticated statistical analysis tools. Their procedures, however, are seldom expressed in algorithmic form, and require human supervision. Of course, this means that they are normally applied off-line.

In [12] trend analysis upon system error logs is applied, trying to predict future hard failures.

In [5] some heuristics, based on the inter-arrival patterns of failures, for fault diagnosis and failure prediction are developed and applied (off-line) to system error logs taken from a large Unix-based file system. Among these heuristics, for example, there is the 2-in-1 rule, which signals a warning when the time of inter-arrival of two failures is less than 1-hour, and the 4-in-1 rule, which fires when four failures occur within a 24-hour.

In [3] a comprehensive and sophisticated methodology for the automatic detection of permanent faults is presented. Errors occurring in a time interval where an higher than normal rate is observed are clustered into "groups". Then the subsequent analysis exploits the quite detailed information given by the error record structure: statistic techniques are recursively used to seek similarities (correlations) among records, that eventually may point to common causes (permanent faults) of each group.

(B) A number of mechanisms, based on simple counting/windowing algorithms implemented in real systems are reported in this group.

In TMR MODIAC, the architecture proposed in [6] and analysed in [7], two failures experienced in two consecutive operating cycles by the same hardware component being part of a redundant structure make the other redundant components to consider it as definitively faulty.

In the IBM machines the idea of thresholding the number of errors accumulated in a time window has long been exploited. In the earlier examples, as in [11], describing the automatic diagnostics of the IBM 3081, the idea is used to trigger the intervention of a maintenance crew; in later models, as in the ES/9000 Model 900 [10], a retry-&-threshold scheme is adopted to cope with transient errors.

In [4] if, after a fault, a channel of the core FTP can be restarted successfully (restoring its internal state from other operating channels), then it is brought back in operation. The value of an associated "health variable", used to differentiate between transient and intermittent failures, is decreased.

In [1] a list of "suspected" processors is generated during the redundant executions, then, a few schemes are suggested for processing this list. Processors participating in the execution of a job and failing to produce a signature matching that of the accepted result, can be taken down immediately for off-line diagnosis, or can be assigned weights, only those with weight exceeding a certain predetermined threshold are then taken down for diagnostics.

As recalled in the Introduction, in [2] the Authors described a mechanism, named α-count, implementing a strategy to decide the point in time when keeping a system component on-line is no longer beneficial. If the rate of detected errors, filtered according to some parameters, exceeds a tunable threshold, the component is flagged for proper treatment. The main point of that paper is that the mechanism can, and in fact has been, modelled and evaluated: its performance is given in terms of specific figures of merit, appropriately introduced. Moreover, procedural actions a designer could take to make good choices for the mechanism's parameters have been sketched. This single-threshold scheme was quite simple, yet well illustrative about the main concepts; the effectiveness of the underlying ideas can be enhanced by straightforward extensions, like that one described in the following Sections.

3 A double-threshold scheme

In [2], all signals related to the correctness of a given component are gathered from any available error detector. Signals feed the filtering mechanism, which is defined in terms of a function α given in the following form:

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

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

Where α is the score associated to each not-yet-removed component *u* to record information about the failures experienced by that component, and $J^{(L)}$ indicates the L-th signal on the generic component u: $J^{(L)} = 1$ corresponding to a failure, $J^{(L)} = 0$ otherwise.

When the value of $\alpha^{(L)}$ exceeds the threshold α_T, the component *u* is considered affected by a permanent or otherwise by a too much frequent intermittent fault; this event produces a *α-signal*, sent to the fault passivation subsystem.

The parameters K and α_T are under the control of the designer. Their values have to be tuned, depending on the system fault rates or probabilities, in order to get the "best" behaviour in terms of some performance figures. In [2] the figures defined have been i) the delay D between an intermittent fault occurrence and its signalling, and ii) the probability HR of (wrongly) identifying a healthy component as faulty (which results into an utilisation shorter than the component's life).

Clearly, low values for both HR and D are conflicting goals: e.g., lowering the threshold value α_T yields faster detection of permanent faults, but increases the probability of unduly junking an unlucky component hit by an occasional storm of transient glitches. The designer is thus faced to arrange a viable trade-off.

To ease this task, let us introduce a two-threshold α-count scheme: now, a component *u* is kept in full service as long as its score α is lower than a *first threshold* α_L. When the value of α grows bigger than α_L, *u* will be restricted to run application programs just for diagnostic purposes: its output will only be gathered by the error signalling subsystem, and its associate α-count mechanism will continue to operate. In other words *u* is considered "suspected" and as such its results are not delivered to the user or relied upon by the system. If α continues to grow, it eventually crosses a *second threshold* α_H, $\alpha_H > \alpha_L$: then the component *u* is diagnosed as affected by a permanent/intermittent fault, and the α-signal is issued, as in the original α-count, to the fault passivation subsystem. When instead α, after some wandering in the (α_H--α_L) "limbo", becomes lower than or equal α_L, then u will be brought back in full service.

With this scheme, the lower threshold can be kept at a low level, to limit the detection delay, while losing not too much on the probability of throwing away good components. Actually, healthy components being hit by occasional fault bursts may be counted off into a non-utilisation period (i.e. until their α-count goes back under the lower threshold), but the probability of irrevocable ousting can be kept as low as desired by means of an appropriately high value for α_H. Another adverse effect, which has to be taken into account in the dimensioning of the system, is due to the necessity of treating the errors showing up in the limbo dwellers. In fact, for continued observation of their behaviour, they need to be nursed exactly as the on-line components, putting their share of burden on the error-processing subsystem.

3.1 The System and Relevant figures of interest

To carry on the modelling and analysis of the double-threshold α-count just introduced, let us recall and update the assumptions established in [2]. The system is considered partitioned into components enclosed in well-defined boundaries, establishing the finest resolution for error detection and fault diagnosis. A *fault passivation subsystem* is in charge of enacting proper actions upon receiving the signals produced by the fault discrimination mechanism. An *error detection subsystem* has the responsibility to judge if a component is deviating from the expected behaviour because of a fault. The error detection results into binary valued signals, which feed the fault discrimination mechanism. After an error occurrence, *error recovery* mechanisms/procedures are available; able to bring back the system into a correct state, where the normal computation is resumed, provided no fault is still hampering the operation.

The goal of our fault discrimination mechanisms may be informally expressed as to try to balance between two conflicting requirements:

i) Signal all components affected by permanent or intermittent faults (referred to in the following as *faulty components*, or *FCs*) as quickly as possible. In fact, gathering information to discriminate between transient and intermittent faults takes time, giving way to a longer latency. These latent faults increase the risk of having to deal with multiple faults, which rises the requirements on the error processing subsystem.
ii) Avoid taking off-line components other than FCs (referred to in the following as *healthy components*, or *HCs*); depriving the system of valid resources may cut on the system usefulness not less than relying on a faltering component.

The performance of the mechanism with respect to the above goal is given quantitative measures by the following figures:

1) The average D of the total elapsed time, after the (permanent or intermittent) fault occurrence in a component u, in which it is maintained in use and relied upon, because its condition has not yet been recognised. D has a slightly different meaning wrt the definition given in [2] (recalled in Sect. 3 above) because it accounts for any time spent in use, even upon a comeback from the "suspected" limbo.

2) A measure of the penalty inflicted by the discrimination mechanism on the utilisation of HCs. The figure HR recalled in Sect 3 from [2] is not fully representative in the double-threshold scheme, since a HC is unused both if wrongly removed from the system and as long as it is kept in the limbo. Therefore a new figure, NU, is adopted, defined as the fraction of component life, (as determined by the expected occurrence of a permanent/intermittent fault) in which the otherwise healthy component *u* is not effectively used in the system.

3.2 Basic assumptions

A generic system component is supposed to behave according to the following assumptions:

1- All fault-related events (error detector triggers, testing routines completion, etc.) occur at discrete points in time; two successive points differ by a (constant) *time unit* (or *step*). For each component, a default "I'm alive and well" *state-indicator* signal is issued at each step, if not overridden by real error signals;
2- The hardware directly supporting the discrimination mechanisms is fault-free;
3- The signal carrying the component's judgement is correct with a probability *c*;
4- System components may be affected by permanent, intermittent or transient faults. During each time unit a component may be affected at most by a single internal fault; therefore, a component experiencing an intermittent fault may also be hit by a transient fault, but not by a permanent fault;
5- Permanent and intermittent faults are exponentially distributed. Therefore the probabilities of their occurrence in a time unit are constant, and will be denoted by q_p and q_i respectively;
6- A component affected by an intermittent fault does repeatedly give rise to an error with a constant probability *q* at each step;
7- A transient fault lasts only for one step. Transient faults are exponentially distributed, with a constant probability of occurrence q_t in a generic time unit.

4 Modelling the double-threshold scheme

To evaluate D and NU, the behaviour of α-count has been modelled by means of two Stochastic Activity Networks (SAN) [8], F_SAN and H_SAN, respectively. In SAN models, such as F_SAN and H_SAN, real valued variables, like α_i, need to be represented by a discrete approximation which lead to approximations on D and NU. However the discrete approximation can be chosen as close to α_i as desired at the price of increasing the time and computational effort necessary to obtain the solution..

4.1 Double-threshold α–count acting on a FC

The behaviour of the double-threshold α–count acting on a FC is modelled using F_SAN (see Figure 1), which allows to compute an upper bound on D, the average number of steps the FC is kept in full service after the permanent/intermittent fault occurrence. Inside D_SAN, α_i is represented by the discrete variable α_inf taken as the value of α_i, truncated to the 4th decimal figure. Being α_inf ≤ α_i, α_inf will then need more steps to reach the thresholds. F_SAN has four places:

i) *not_remov* (initially set to 1): the presence of a token in this place means that u is not yet flagged as faulty;
ii) *remov*: a token here marks the actual identification of u as FC, with the attending emission of the α-signal;
iii) *alpha_state*: it maintains the current value of α_inf (the lower discrete approximation chosen for α_i);
iv) *susp* (initially set to 0): it represents the alternation of periods in which the component is "suspected" (one token in *susp*) with those where the component is used and provides results to the user (no tokens in *susp*).

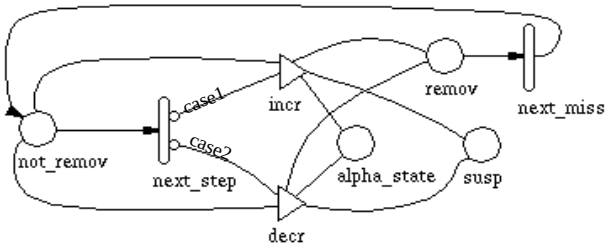

Figure 1. F_SAN: Double-threshold α–count acting on a Faulty Component

The two output-gates *incr* and *decr* perform the basic steps for computing α_inf. When α_inf reaches the value α_L the gate *incr* sets to 1 the marking of *susp*. When instead α_inf reaches the value α_H, *incr* deposits a token in the place *remov* and resets to 0 the marking of *alpha_state* and *susp* for a proper restart in *next_miss*. When the value of α_inf becomes smaller than the α_L threshold, the gate *decr* has the task of setting to 0 the number of tokens in *susp*. Gates *incr* and *decr* take care of properly putting tokens in *not_remov* to keep things running.

The exponentially distributed timed activity (with rate 1) *next_step* represents the receipt of a signal; it has two cases: *case 1*, representing the occurrence of $J^{(L)} = 1$, and *case 2*, representing that of $J^{(L)} = 0$. The probability associated to *case 1* is obtained as the probability of two disjoint events:

(1) a failure of u_i (with probability $q+q_t$ that is either an activation of the intermittent or an occurrence of a transient fault – we neglect the trivial case of permanent faults) properly detected (with probability c); or

(2) a success of the component ($1-q-q_t$) wrongly detected ($1-c$).

In the same way one can compute the probability for *case 2*. Thus:

$$P(\text{Case 1}) = (q+q_t)c + (1-q-q_t)(1-c)$$
$$P(\text{Case 2}) = (q+q_t)(1-c) + (1-q-q_t)c$$

The steady state probability of "#not_remov=1" divided by the steady state probability of "#remov=1" gives the number of iterations to the complete removal of the unit. The steady state probability of "#not_remov=1 and #susp=0" divided by the steady state probability of "#not_remov=1": gives the fraction of the full utilisation of the unit. D is then obtained by multiplying these two factors:

$$D = \frac{P(\#\,not\_remov = 1 \text{ AND } \#\,susp = 0)}{P(\#\,not\_remov = 1)} \bullet \bullet \frac{P(\#\,not\_remov = 1)}{P(\#\,remov = 1)} = = \frac{P(\#\,not\_remov = 1 \text{ AND } \#\,susp = 0)}{P(\#\,remov = 1)}$$

.

4.1.2 Double-threshold α–count acting on a HC

H_SAN, in Figure 2, models the behaviour of the double-threshold α–count acting on a HC. α_i is represented here by the discrete variable α_sup, chosen as the nearest four-decimal figure value greater than α_i. It allows the computation of an upper bound of NU, the utilisation loss of a healthy component.

H_SAN differs from F_SAN because of

i) The presence of the *init* gate, which is in charge of the proper restart of the SAN cycle,

ii) *alpha_state*: maintains the current value of an higher approximation α_sup to α_i; and

iii) The activity *next_step* has now three cases: *case 1* represents the occurrence of a permanent or intermittent fault in u, which thus becomes a FC, *case 2* and *case 3* represent respectively the receipt of $J^{(L)} = 1$ and of $J^{(L)} = 0$ when u is a HC.

$$P(\text{Case 1}) = q_p + q_i$$
$$P(\text{Case 2}) = q_t c + (1-q_p-q_i-q_t)(1-c)$$
$$P(\text{Case 3}) = q_t(1-c) + (1-q_p-q_i-q_t)c$$

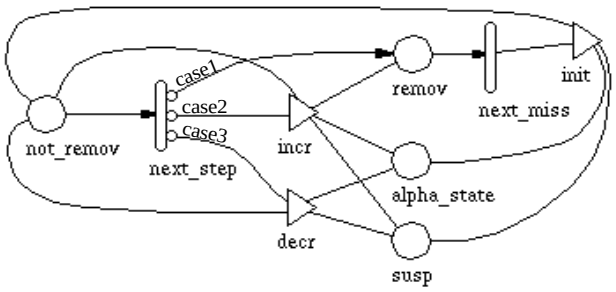

Figure 2. H_SAN: Double-threshold α–count acting on a Healthy Component.

From its definition, NU is computed as the difference between the number of steps to the expected occurrence of a permanent/intermittent fault (E_life) and the expected number of steps in which the component is utilised by the system (E_util) divided by the former. In this model, E_life is given by: $E\_life = \frac{1}{(q_i + q_p)}$ and E_util is the expected number of steps in which the HC is not suspected, i.e., used and relied upon. E_util is determined as the steady state probability of "#not_remov=1 and #susp=0" divided by the steady state probability of "#not_remov=0":

$$E\_util = \frac{P(\#\,not\_remov = 1 \text{ AND } \#\,susp = 0)}{P(\#\,not\_remov = 0)}$$

.

Then:

$$NU = \frac{\frac{1}{(q_i + q_p)} - \frac{P(\#\,not\_remov = 1 \text{ AND } \#\,susp = 0)}{P(\#\,not\_remov = 0)}}{\frac{1}{(q_i + q_p)}} = 1 - \left(\frac{P(\#\,not\_remov = 1 \text{ AND } \#\,susp = 0)}{P(\#\,not\_remov = 0)} \bullet (q_i + q_p)\right)$$

UltraSAN [9] has been used to analytically solve the SAN models in Figures 1 and 2 and to evaluate D and NU.

Symbol	Definition	Value
q_t	probability of transient fault per time-unit	10^{-5}
$q_p + q_i$	probability of intermittent or permanent fault per time unit	10^{-6}
q	probability of activation of an intermittent fault per time unit, given that the component is affected by an intermittent fault	0.1
c	probability that a signal on the components behaviour is correct	$1\text{-}10^{-5}$

K	the ratio by which α is decreased upon receiving $J^{(L)}=0$	
α_T	threshold for identifying a component as affected by a permanent or intermittent fault in the single scheme [2]	
α_H	higher threshold for identifying a component as affected by a permanent or intermittent fault in the double-threshold scheme	
α_L	lower threshold keeping a component in full service in the double-threshold scheme	

Table 1. Symbols, definitions and default values

Table 1 recalls the symbols used for all the parameters, with their definition and, for some of them, the default values used in all the plots. The values for q_t, $q_p + q_i$ are derived assuming that the probability of intermittent or permanent fault be 10^{-4}/hour, that the probability of transient faults be 10^{-3}/hour, and that time be discretized into 1/100 hour units.

5.1 Comparisons between single and double threshold α-count

To discern the added features of the double-threshold scheme over the original single-threshold α-count, the performance figures resulting from the application of the single-threshold α-count are compared with those obtainable by the double-threshold variant.

Figure 3 shows the performance figures D and NU for the single-threshold α-count with α_T=2 and K varying in the range [0.85, 0.999]. It also contains some curves obtained for the double-threshold α-count with proper settings of α_L and α_H which demonstrate better performance in almost all the range of K. In particular for K in the range [.94, .998] the double-threshold with α_L = 0.8 and α_H = 3 has better D and lower NU than the single-threshold. This better behaviour is also obtained by setting α_L = 0.4 and α_H = 2.5 in the wider range [.86, .997]. It must be pointed out that, despite not all the range is covered, the settings chosen for the double-threshold α-count allow the scheme to outperform the single-threshold one in the most interesting interval of K where better absolute performances are observed.

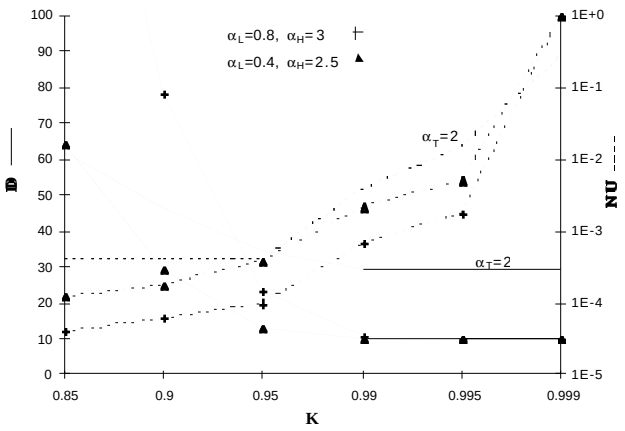

Figure 3: Comparisons of single and double threshold α-count at varying K.

Figure 4 turns the attention to the other parameter of the single-threshold α-count: α_T. The plots of D and NU are given for single-threshold α-count with K fixed to .95 while α_T varies in the range [1.5, 2.5]. Again the Figure shows also curves obtained for the double-threshold α-count with proper settings of α_L and α_H which demonstrate better performance. In this case, the setting α_L = 1.1 and α_H = 2.7 has better D and lower NU than the single-threshold for values of α_T greater than 1.9. The setting α_L = 0.8 and α_H = 3.0 allows better performances when α_T is the in the range [1.55, 2.05].

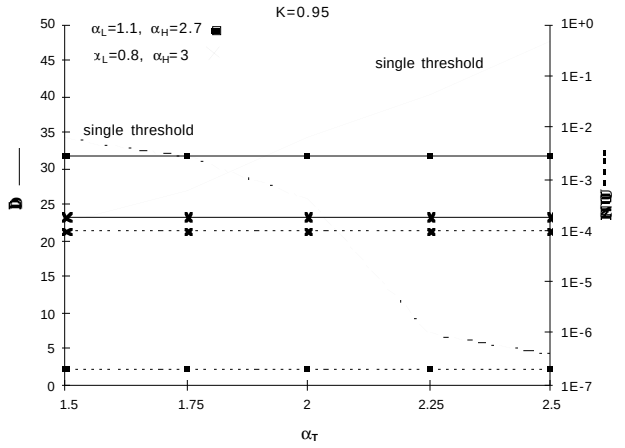

Figure 4: Comparisons of single and double threshold α-count at varying α_T.

Overall, the Figures show that, for all values of K with fixed α_T or for all values of α_T with fixed K, combinations of α_L and α_H exist which allow the double-threshold α-count outperform the single-threshold mechanism. The double-threshold α-count shows either better D with at least the same NU or better NU with no worse D. A proper setting for the parameters α_H and α_L has to be

chosen and, obviously, the range [α_L, α_H] always encompasses the value of α_T.

5.2 Tuning of the double threshold α-count

Now consider what could be the typical procedure a system designer would follow to set up the internal parameters of the mechanism. Low values of NU and of D being still conflicting objectives, the proper tuning of parameters requires the definition of their relative importance. More generally, tuning of parameters for a specific system can be done once the system designer has given constraints on the desired behaviour of the mechanism. Here we show the steps needed to set the parameters to accomplish the requirement "D must be optimised while NU must take values lower than a $5\ 10^{-3}$".

To tune the parameters in the single-threshold case, [2] has shown that, given the external parameters, i.e. the fault rates and their characterisations, a near-optimal value of K can be determined. Actually, Figure 3 above suggests the same also for the double-threshold α-count. In fact, increasing values of K improve D (which gets lower values); this improvement is much faster for low values of K up to a region around a value K_D (.99 in case of Figure 3), becoming evanescent for higher values. A similar behaviour is observed for NU. NU grows slowly for low values of K up to a value K_{NU} (.95 in case of Figure 3), whereas it becomes worse and worse for higher values. It appears thus that values for K should be chosen between K_D and K_{NU}.

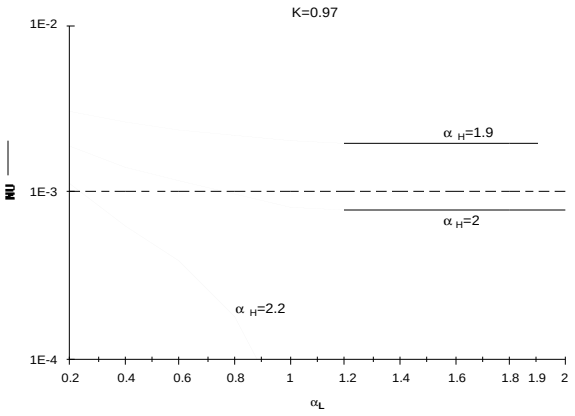

Figure 5: Values of NU as a function of α_H for varying α_L

Having set a value for K leading to a near optimal tuning (.97 in our case), proper values for α_H and α_L can be derived as follows. Figure 5 shows plots of NU for different values of for α_H at varying α_L. It shows that: 1) α_H =1.9 does not allow to reach the target NU for any value of α_L; 2) with α_H =2, values of α_L higher than 0.8 are required; 3) with α_H =2.2 or higher the target NU can be reached with values of α_L higher than 0.25. In this way the combinations of these parameters which fit the target NU are identified.

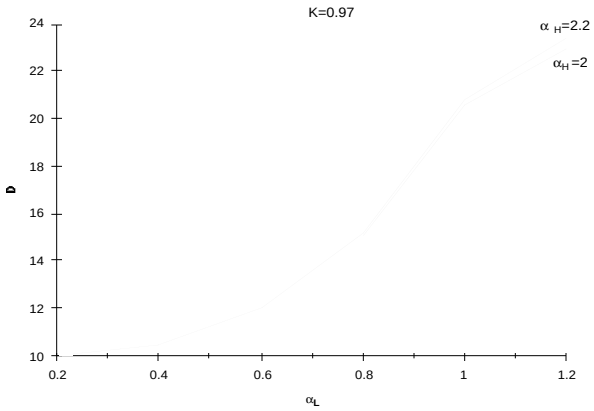

Figure 6: Values of D as a function of α_H for varying α_L (for the combinations satisfying the target NU)

Then, the values to be assigned to α_H and α_L can be chosen by inspecting the plots of Figure 6. Here D is computed for the admitted combinations of α_H and α_L. Actually, the curves related to α_H =2 and α_H =2.2 almost overlap in a wide range of α_L up to α_L=0.8. The reason is the following. To have different values of D with the same α_L, α_i must reach a value between 2 and 2.2 so that with α_H =2.2 the unit is maintained in the system while in the other it is taken off-line. Moreover, the unit has then to experience enough successes to bring back its α_i to a value lower than α_L. This is the only way D may further increase. However, with the fault rates chosen in this example, this event is very unlikely for values of α_L lower than 0.8. From Figure 6, the obvious choice is to select α_L =0.25 and α_H =2.2 or higher.

6 Conclusions

A new mechanism, belonging to the count and threshold family α-count, has been introduced in this paper to discriminate intermittent and permanent faults against low rate, low persistency transient faults. The double-threshold α-count, as a member of this family, is characterised by:

a) Tunability through internal parameters, to warrant wide adaptability to a variety of system requirements;
b) Generality with respect to the system context, in which they are intended to operate, to ensure wide applicability;

c) Simplicity of operation to allow exhaustive analysability through simple models.

The behaviour of the double-threshold α-count has been quantitatively analysed, in terms of:

1) Total elapsed time, after a (permanent or intermittent) fault occurrence, in which a component is maintained in use and relied upon;
2) A measure of the penalty inflicted by the discrimination mechanism on the utilisation of healthy components.

It has been shown that by a proper setting of the internal parameters the new scheme outperforms the single-threshold one. Moreover the paper includes indications on how to tune the parameters to obtain a good behaviour. The design procedure retains the simplicity of that already shown for the single-threshold, with the added bonus of granting the designer more freedom in the parameter values choice.

Acknowledgement

This work has been partially supported by the CEC in the framework of the ESPRIT Project 20716 GUARDS, Generic Upgradable Architecture for Real-Time Dependable Systems.

References

[1] P. Agrawal, "Fault Tolerance in Multiprocessor Systems without Dedicated Redundancy," IEEE Transactions on Computers, Vol. C-37, pp. 358-362, 1988.

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

[3] R. K. Iyer, L. T. Young and P. V. K. Iyer, "Automatic Recognition of Intermittent Failures: An Experimental Study of Field Data," IEEE Transactions on Computers, Vol. C-39, pp. 525-537, 1990.

[4] J. H. Lala and L. S. Alger, "Hardware and Software Fault Tolerance: A Unified Architectural Approach," in Proc. 18th International Symposium on Fault-Tolerant Computing, Tokyo, Japan, 1988, pp. 240-245.

[5] T. -T. Y. Lin and D. P. Siewiorek, "Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis," IEEE Transactions on Reliability, Vol. 39, pp. 419-432, 1990.

[6] G. Mongardi, "Dependable Computing for Railway Control Systems," in Proc. DCCA-3, Mondello, Italy, 1993, pp. 255-277.

[7] M. Nelli, A. Bondavalli and L. Simoncini, "Dependability Modelling and Analysis of Complex Control Systems: an Application to Railway Interlocking," in Proc. EDCC-2 European Dependable Computing Conference, Taormina, Italy, 1996, pp. 93-110.

[8] W. H. Sanders and J. F. Meyer, "A Unified Approach for Specifying Measures of Performance, Dependability and Performability," in "Dependable Computing for Critical Applications, Vol. 4: of Dependable Computing and Fault-Tolerant Systems", A. Avizienis and J. Laprie Ed., Springer-Verlag, 1991, pp. 215-237.

[9] W. H. Sanders, W. D. Obal, M. A. Qureshi and F. K. Widjanarko, "The UltraSAN Modeling Environment," Performance Evaluation Journal, special issue on Performance Modeling Tools, Vol. 24, pp. 89-115, 1995.

[10] L. Spainhower, J. Isenberg, R. Chillarege and J. Berding, "Design for Fault-Tolerance in System ES/9000 Model 900," in Proc. 22nd International Symposium on Fault-Tolerant Computing, Boston, Massachusetts, USA, 1992, pp. 38-47.

[11] N. N. Tendolkar and R. L. Swann, "Automated Diagnostic Methodology for the IBM 3081 Processor Complex," IBM J. Res. Develop., Vol. 26, pp. 78-88, 1982.

[12] M. M. Tsao and D. P. Siewiorek, "Trend Analysis on System Error Files," in Proc. 13th International Symposium on Fault-Tolerant Computing, Milano, Italy, 1983, pp. 116-119.