Proc. of the 11th International Symposium on Fault Tolerant Computing ,
June 1981.

# A MODULE REPLACEMENT POLICY FOR DYNAMIC REDUNDANCY FAULT-TOLERANT COMPUTING SYSTEMS

Israel Koren

Department of Electrical Engineering
Technion-Israel Institute of Technology
Haifa, Israel

Menachem Berg

Department of Quantitative Methods
College of Business Administration
University of Illinois at Chicago Circle
Chicago, USA

## Abstract

Fault-tolerant computing systems are designed to provide an error-free operation in the presence of faults. The system recovers from the effects of a fault by employing certain recovery procedures like program rollback, reload and restart, etc. These recovery procedures, as efficient as they may be, result in interruptions in the system's operation. Interruptions which reduce the computational availability of the system and may have an intolerable impact on the performance of a critical real-time control system. Fault-tolerant systems for critical applications include standby spares that are ready to replace active modules which fail to recover from the effects of a fault. A standby spare may also be used to replace a module suffering from frequent fault occurrences resulting in too many applications of the recovery process, in order to increase the computational availability. In this case a module replacement policy is needed. This policy will indicate upon occurrence of a module failure whether to retry the module or replace it considering mission time, probability of a system crash and computational availability. A module replacement policy for dynamic redundancy systems is presented in this paper and the improvement in computational availability due to its application is illustrated.

## I. Introduction

The ultimate goal when incorporating fault-tolerance into a computing system is to have an operational system throughout the mission time (in the presence of faults) with as few interruptions as possible. An operational computing system is not just a system producing error-free outputs but useful error-free outputs. Any time the system is performing tasks which are not directly related to its operational objectives, this time period is considered wasted. Examples of such tasks are error detection procedures, recoveries from failures, restoring of data or programs, reconfigurations of the system, initialization and synchroniztion processes. Consequently, the computational availability[3] rather than system availability should be maximized. The difference between the two measures may be quite substantial, e.g., a frequently occurring failure in an active module of a computing system may result in a large number of time-consuming error-recoveries and resynchronizing processes. A situation like this might have an intolerable impact on the performance of a critical real-time control system (e.g., in aerospace applications like vehicle guidance systems). In these cases it might be beneficial to replace such a module with a spare one if available, in order to achieve higher computational availability. To study these phenomena we have to consider the possible failure modes and the appropriate recovery procedures. Failures occurring in digital systems come from a variety of sources, e.g., defective components, design deficiencies and environmental causes. Each of these sources generates faults in a different way and of different types. However, the faults may be roughly classified into three types, permanent faults, intermittent faults and transient faults[1,2,10,14-17,19]. Permanent faults are solid hardware failures, intermittent faults are due to physical defects in the hardware that manifest themselves intermittently in an unpredictable manner (in this class we may include data sensitive design errors or "software failures"[5]). Transient faults are due to temporary environmental conditions (such as temperature, humidity, vibration, power fluctuation, electromagnetic fields, etc.) which may persist for an unpredictable period of time[14]. Although intermittent and transient faults disappear after a short duration, their damage to the information structure of the system is in many cases permanent[15-16,21]. Since the exact nature of the fault and its effects are usually unknown upon occurrence, recovering from the fault is one of the system's most complex and difficult functions, e.g., [8,20]. If the system suffers a permanent fault then the failing module must be identified and replaced. However, it is known that non-permanent faults (i.e., intermittent and transient faults) constitute the majority of the faults occurring in computing systems[2,17]. Hence, recovery capabilities for non-permanent faults which do not require giving up hardware resources prematurely are usually built into these systems. Various recovery techniques are used nowadays in fault-tolerant computing systems like instruction retry, program roll-back, reload and restart, etc.[16,17]. None of these techniques can be effective against all possible faults. Hence, several recovery steps are employed in support of one another to increase effectiveness[16,17]. Due to the complexity of the recovery problem each recovery procedure has its deficiencies and may fail and cause a system crash[17,20]. In some systems recovery deficiencies account for as much as 35% of the downtime[20]. If the recovery procedure is successful an overhead time is involved. First there is a detection time between fault occurrence and fault detection, then there is the recovery time including initialization and

synchronization of the failing module[7-8,15-16]. Usually, the information processing performed by the system during the detection time (and even before that if a reload and restart operation is needed) is contaminated and must be repeated. Thus, a cost to the system is associated with every recovery procedure activated. The cost of a single application of a recovery procedure might be low however, a frequently occurring non-permanent fault will considerably reduce the computational availability of the system. Moreover, the rate of some intermittent faults like those caused by deteriorating or aging components, gradually increases until they become permanent. Their transition into solid faults may take from a few minutes to several months during which the frequency of their occurrence increases intolerably. Such a situation should be avoided on time.

In addition, too many applications of a recovery procedure will undoubtedly increase the probability of an unsuccessful recovery leading to a system failure. Consequently, in some cases it may be worthwhile to switch out the module which is subject to non-permanent fault occurrences and reconfigure the system. On the other hand, replacing an active module by a spare whenever a few non-permanent faults occur is not recommended since the premature retiring of a module which might still have a useful life service will unnecessarily reduce the system's mission time. The possibility of a system failure during reconfiguration due to imperfect coverage (e.g., a switching failure) and the overhead time associated with reconfiguration must also be considered.

What is needed is a module replacement policy that will indicate the kind of operation to be taken by the system whenever a module fails, i.e., should we retry the module that has failed or maybe switch it out and replace it by a spare considering computational availability, mission time and the probability of a system failure. Clearly, the optimal module replacement policies for various structures of fault-tolerant computing systems are not necessarily the same. In the following we introduce a replacement policy for dynamic (stand-by) redundancy fault-tolerant systems. Similar policies for other configurarions of fault-tolerant systems are now being investigated.

## II Dynamic Redundancy Fault-Tolerant Systems

Consider a computing system, one of whose modules has $m$ standby spares ready to be switched in upon failure of the active module. The active module is subject to fault occurrences and we adopt here the viewpoint that fault occurrences obey a Poisson process[4-5,7-13,15-19]. Although the failure rate is treated as a constant in most related works, it is recognized that it may be a function of time and other module parameters. It is apparent, in particular, that as time goes on certain types of faults are more likely to occur and the status of the module may be deteriorating. One way to model the deteriorating status of the module is to assume that the failure rate increases with every fault occurrence. Thus, when a non-permanent fault occurs frequently the increasing failure rate will reflect the true status of the module. Another way is to assume that the failure rate increases only when the time between two successive occurrences is less than some prespecified time period. We adopt in the following the first alternative due to its relative simplicity. Thus, a series of non-decreasing failure rates results, i.e., $\lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_k$. It is further assumed that at some point, say after $K$ failures the failure rate does not increase any more.

Suppose a module which was subject to repetitive non-permanent fault occurrences has failed, the decision whether to retry it or replace it depends among other factors, upon the module's failure rate and system factors like the number of yet available spares and the remaining time to completion of the prespecified mission. If the failure rate is high it would be desirable to replace the module in order to reduce the probability of a future failure. This however, should be weighed against the risk of spare exhaustion before the mission is over. For a given mission time $t$, number of spare modules $m$ and module failure rate $\lambda$ we will calculate a time instant $t_1$. Until we reach this time instant we keep on retrying the module whenever it fails as long as its failure does not become permanent. Once we reach $t_1$ we will replace the module upon failure since we have enough spares for the remaining mission time and the risk of spare exhaustion is low. To find $t_1$ we derive an expression for the cost of the computational down time periods that the system will suffer throughout its mission time. In computational down time we include also time periods through which the system is operational but is not doing any useful computation directly related to its mission. This module replacement policy is depicted in Fig. 1 where some arcs are marked with probabilities which are defined in the following.

According to our policy, if the active module with a failure rate $\lambda_k$ fails at time $u > t_1$ then it is replaced without retrying it. Let $\theta_1$ denote the time needed to switch out the failed module, switch in a good spare, initialize the spare (restart the program) and synchronize it. The cost associated with the replacement is $C_1 = C \cdot \theta_1$ where $C$ is the cost of system down time per time unit. While replacing the failing module by a good spare the system might crash and we denote the probability of a system failure by

$r = Pr$ {the system fails due to unsuccessful replacement/a module failed}.

With probability (1-r) the replacement will be successful, the new active module will have a lower failure rate $\lambda_0$, a smaller number of spares, i.e., m-1 and residual mission time $t-u$.

If the failure occurs before $t_1$ we try to recover from the effects of the fault and resume normal operation without replacing the module. Various fault recovery techniques like instruction retry, program rollback, memory copy, system restart and others are used nowadays[15-17]. They differ widely in their effectiveness against specific fault types and in the amount of time needed for the recovery process. Some faults may be recovered from by merely instruction retry while others require more

time consuming recovery techniques. Since the severity of the occurring fault is unknown the most appropriate recovery technique cannot be selected. Hence, a multi-phased recovery process should be planned[16-17]. In the first phases of such a process simple and high-speed recovery techniques are employed while complex and time-consuming recovery techniques are left for the last phases. Consequently, the actual time used for recovery is a random variable and we are interested in its expected value denoted by

$$\theta_2 = E\{\text{total computation time lost due to recovery}\}.$$

When executing the recovery process there are three possible outcomes. The first - the failing module has fully recovered and is now operational. The second - all recovery techniques employed failed, the fault is considered permanent and a replacement is taking place. And the third - the system is unable to recover and crashes. To incorporate these possibilities we denote

$p = \text{Prob}\{\text{the fault is permanent/a module failed}\}.$

$s = \text{Prob}\{\text{the system fails due to inability to recover/a module failed}\}.$

(Note that $1-s$ is the coverage probability[4,18].)

We associate with the recovery process a cost $C_2$ which accounts for the computational system down time and is given by

$$C_2 = C \cdot \theta_2.$$

The cost $C_2$ is not necessarily linear in $\theta_2$. One can envision a critical control application where a brief down time period may be tolerated while a longer one may be intolerable and hence very costly. However, as a first approximation we use a linear function.

If the system crashes at time $u$ then $t-u$ is the residual mission time, a cost of $C_3 \cdot (t-u)$ is incurred where $C_3$ is the cost of system down time per time unit and $C_3 \geq C$ since total system failure is more costly than a short break in the system's operation.

To find the optimal time instant $t_1$ we derive an expression for the expected cost of the system's computational down time and minimize it with respect to $t_1$.

The expected cost is denoted by $C(k,m,t,t_1)$ where $k$ is the total number of faults that have occured so far in the active module. For simplicity, we denote

$$C(k,m,t) = \min_{0 \leq t_1 \leq t} C(k,m,t,t_1).$$

This cost function is calculated using the following recursive relation which is derived from Fig. 1.

$$\begin{aligned} C(k,m,t) = \min_{0 \leq t_1 \leq t} \Big\{ & \int_0^{t_1} g_k(u)[s \cdot C_3 \cdot (t-u) \\ & + (1-s)C_2 \cdot ((1-p) + p(1-r)) \\ & \qquad + (1-s)(1-p)C(k+1,m,t-u) \\ & + (1-s)p(r \cdot C_3 \cdot (t-u) + (1-r)C_1 \\ & \qquad + (1-r)C(0,m-1,t-u))]du \\ & + \int_{t_1}^{t} g_k(u)[r \cdot C_3 \cdot (t-u) + (1-r)C_1 \\ & \qquad + (1-r)C(0,m-1,t-u)]du \Big\} \end{aligned} \quad (1)$$

where $g_k(u) = \lambda_k e^{-\lambda_k u}$ is the failure probability density function. Our assumption that $\lambda_{k+1} = \lambda_k$ for any $k \geq K$ results in the following relation

$$C(k+1,m,t) = C(k,m,t) \quad \text{for } k \geq K. \quad (2)$$

Hence, the boundary value $C(K,m,t)$ is calculated first and its calculation is straightforward. Then, the values of the other $C(k,m,t)$ may be calculated using (1).

In the special case where no standby spares are available ($m = 0$), $t_1 = t$ and equation (1) transforms into

$$C(k,0,t) = \int_0^t g_k(u)\Big[\alpha C_2 + \alpha C(k+1,0,t-u) + (1-\alpha)C_3 \cdot (t-u)\Big]du \quad (3)$$

where $\alpha = (1-s)(1-p)$.

As an example we consider a system with a single standby spare where each module has one out of two possible failure rates, i.e., $\lambda_0$ is the initial failure rate and $\lambda_1$ is the failure rate after the first fault occurrence (and successful recovery), from that time on the failure rate stays constant. Here we have to calculate four values of $C(k,m,t)$, namely $C(0,1,t)$, $C(1,1,t)$, $C(0,0,t)$ and $C(1,0,t)$. The boundary value $C(1,0,t)$ is calculated directly yielding

$$C(1,0,t) = C_3 \cdot t - \frac{\frac{C_3}{\lambda_1} - \alpha C_2}{1-\alpha}\left(1 - e^{-(1-\alpha)\lambda_1 t}\right). \quad (4)$$

Substituting (4) in (3) we obtain

$$C(0,0,t) = C_3 t + P\left(1 - e^{-\lambda_0 t}\right) - Q\left(1 - e^{-(1-\alpha)\lambda_1 t}\right) \quad (5)$$

where

$$P = \frac{\lambda_1 - \lambda_0}{[\lambda_0 - (1-\alpha)\lambda_1]}\left[(1-\alpha)\frac{C_3}{\lambda_0} - \alpha C_2\right]$$

$$Q = \frac{\alpha \lambda_0}{(1-\alpha)[\lambda_0 - (1-\alpha)\lambda_1]}\left(\frac{C_3}{\lambda_1} - \alpha C_2\right).$$

$C(1,1,t)$ is again calculated directly yielding

$$C(1,1,t) = \min_{0\le t_1\le t} C(1,1,t,t_1)$$

$$= \min_{0\le t_1\le t} \Big\{ C_3\cdot t + A\cdot\Big(1-e^{-(1-\alpha)\lambda_1 t_1}\Big) + Be^{-\lambda_o t} + D\cdot e^{-(1-\alpha)\lambda_1 t_1}\Big(1-e^{-\lambda_1 t_2}\Big) + Ee^{-\lambda_1 t_2}e^{-(1-\alpha)\lambda_1 t_1} - Fe^{-\lambda_o t_2}e^{-(1-\alpha)\lambda_1 t_1} + G\cdot e^{-(1-\alpha)\lambda_1 t_1}\Big\} \qquad (6)$$

where

$$A = \frac{1}{(1-\alpha)}\left[\beta(C_1+C_2+P-Q)+\left(\alpha C_2-\frac{C_3}{\lambda_1}\right)\right]$$

$$B = \frac{\beta\cdot\lambda_1\cdot P}{[\lambda_o-(1-\alpha)\lambda_1]} \quad ; \quad D = \left[(1-r)(C_1+P-Q) - \frac{C_3}{\lambda_1}\right]$$

$$E = (1-r)\left[\frac{\lambda_1}{\lambda_1-\lambda_o}P - \frac{1}{\alpha}Q\right]; \quad F = \left[\frac{\lambda_1\beta}{\lambda_o-(1-\alpha)\lambda_1} + \frac{(1-r)\lambda_1}{\lambda_1-\lambda_o}\right]P$$

$$G = \left[\lambda_1\beta\cdot t_1 + \frac{1-r}{\alpha}\right]Q \; ; \; \beta = p(1-s)(1-r) \text{ and } t_2 = t-t_1.$$

C(0,1,t) was calculated numerically using a programmable calculator and is not reproduced here. The values of the four cost functions were computed for various values of the system's parameters and some of the results are illustrated in Figs. 2, 3 and 4. In Fig. 2 the dependence of $C(0,1,t,t_1)$ and $C(1,1,t,t_1)$ on the policy parameter $t_1$ is depicted in three cases for a mission time $\lambda_o t = 4$. Both cost functions are minimized for $t_1 = 0$, $\lambda_o t_1 = 2.7 = 0.675\lambda_o t$ and $t_1 = t$ in cases (a), (b) and (c), respectively. The reason for having the same optimal value of $t_1$ for both functions being the fact that the failure rate changes from $\lambda_o$ to $\lambda_1$ upon the first fault occurrence. Note that in case (a) the optimal value of $t_1$ is zero and the corresponding cost C(0,1,4) is 1506. If instead of applying the optimal policy which calls for module replacement upon failure, we keep on retrying the module until it either fails permanently or causes a system failure; the expected cost increases by 14.6% to 1726.

In Fig. 3 the four cost functions are plotted for two different cases (i) and (ii). By comparing C(0,1,t) to C(0,0,t) or in general C(0,m,t) to C(0,m+1,t) we obtain the cost gain due to an additional spare module. Thus, by comparing the gain C(0,m+1,t) - C(0,m,t) to the cost of a spare module we may determine the benefit of adding a spare.

To analyze the effectiveness of the module switching policy we have calculated the improvement in the cost function due to the optimal selection of $t_1$ in the two cases shown in Fig. 3. The results are illustrated in Fig.4. In case (i) the selection of the right switching instant $t_1$ is crucial for relatively short mission times in contrast to case (ii). Such a result is expected since in (i) where the optimal policy calls for early replacement it is beneficial for a short mission time when we still have a standby spare. For longer mission times it is less beneficial since the probability that we still have a spare is very low. In case (ii) the percentage of improvement is highest for mission times around $\lambda_o t = 4$ because by retrying the active module and not replacing it too soon we increase the probability of system survival.

## III Conclusions

The idea of a module replacement policy has been introduced and one such policy has been developed for dynamic redundancy fault-tolerant systems. We have illustrated that the application of a module replacement policy might improve substantially the computational availability of a fault-tolerant system. However, further research steps have to be carried out before the importance of module switching policies and the feasibility of their practical implementation can be established. Furthermore, separate module replacement policies have to be developed for other fault-tolerant structures like hybrid redundancy systems and gracefully degrading systems. Hybrid systems provide a faster and more reliable fault diagnosis and have a short term fault masking capability. Consequently, the overhead associated with recovery and replacement procedures in hybrid systems is less than in dynamic systems however, it is not eliminated since resynchronization is still required. The need for module switching policies in gracefully degrading systems is even higher. These systems are designed to provide a high grade of service by reconfiguring the system whenever a module fails permanently. Here, if a module is subject to frequent non-permanent fault occurrences we have to decide whether to switch it out and degrade the system capacity or keep on retrying it. In addition to the already mentioned system parameters we have in this case to take into account also the degraded computational capacity of the system.

## References


1. A. Avizienis, "Architecture of Fault-Tolerant Computing Systems", Digest of the Fifth International Symposium on Fault-Tolerant Computing, pp. 3-16, June 1975.
2. M. Ball and F. Hardie, "Effects and Detection of Intermittent Failures in Digital Systems", AFIPS Conference Proceedings, Vol. 35, pp. 329-335,FJCC 1969.
3. M.D. Beaudry, "Performance-Related Reliability Measures for Computing Systems", IEEE Trans. on Computers, Vol. C-27, pp. 540-547, June 1978.
4. W.G. Bouricius, W.C. Carter, D.C. Jessep, P.R. Schneider and A.B. Wadia, "Reliability Modeling for Fault-Tolerant Computers", IEEE Trans. on Computers, Vol. C-20, pp. 1306-1311, Nov. 1971.
5. A. Costes, C. Landrault, and J.C. Laprie, "Reliability and Availability Models for Maintained Systems Featuring Hardware Failures and Design Faults", IEEE Trans. on Computers, Vol. C-27, pp. 548-559, June 1978.
6. D. Davies and J.F. Wakerly, "Synchronization and Matching in Redundant Systems", IEEE Trans. on Computers, Vol.C-27, pp. 531-539, June 1978.

References (cont'd)

7. A.L. Hopkins, Jr., T.B. Smith, III, and J.H. Lala, "FTMP - A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft", Proc. of the IEEE, Vol. 66, pp. 1221-1239, Oct. 1978.

8. H. Ihara et al., "Fault-Tolerant Computer System with Three Symmetric Computers," Proc. of the IEEE, Vol.66, pp. 1160-1177, Oct. 1978.

9. A.D. Ingle and D.P. Siewiorek, "A Reliability Model for Various Switch Designs in Hybrid Redundancy", IEEE Trans. on Computers, Vol.C-25, pp. 115-133, Feb. 1976.

10. I. Koren and S.Y.H. Su, "Reliability Analysis of N-Modular Redundancy Systems with Intermittent and Permanent Faults", IEEE Trans. on Computers, Vol. C-28, pp. 514-520, July 1979.

11. J.H. Lala and A.L. Hopkins, Jr., "Survival and Dispatch Probability Models of the FTMP", Digest of the Eighth International Symposium on Fault-Tolerant Computing, pp. 37-43, June 1978.

12. J. Losq, "A Highly Efficient Redundancy Scheme: Self-Purging Redundancy", IEEE Trans. on Computers, Vol. C-25, pp. 569-578, June 1976.

13. J. Losq, "Influence of Fault-Tolerant and Switching Mechanisms on the Reliability of Stand-by Systems", Digest of the Fifth International Symposium on Fault-Tolerant Computing, pp. 81-86, June 1975.

14. S.R. McConnel, D.P. Siewiorek, and M.M. Tsao, "The Measurement and Analysis of Transient Errors in Digital Computer Systems", Digest of the Ninth International Symposium on Fault-Tolerant Computing, pp. 67-69, June 1979.

15. P.M. Merryman and A. Avizienis, "Modelling Transient Faults in TMR Computer Systems", Proc. 1975 Annual Reliability and Maintainability Symposium, Washington, D.C., pp. 333-339, January 1975.

16. Y.W. Ng and A. Avizienis, "A Model for Transient and Permanent Fault Recovery in Closed Fault-Tolerant Systems", Digest of the Sixth International Symposium on Fault-Tolerant Computing, pp. 182-188, June 1976.

17. D.P. Siewiorek et al., "A Case Study of C.mmp, Cm*, and C.vmp: Part I - Experiences with Fault Tolerance in Multiprocessor Systems", Proc of the IEEE, Vol. 66, pp. 1178-1199, Oct. 1978.

18. J.J. Stiffler, "The Reliability of a Fault-Tolerant Configuration Having Variable Coverage", IEEE Trans. on Computers, Vol. C-27, pp. 1195-1197, Dec. 1978.

19. S.Y.H. Su, I. Koren, and Y.K. Malaiya, "A Continuous Parameter Markov Model and Detection Procedures for Intermittent Faults", IEEE Trans. on Computers, Vol. C-27, pp. 567-570, June 1978.

20. W.N. Toy, "Fault-Tolerant Design of Local ESS Processors", Proc. of the IEEE, Vol. 66, pp. 1126-1145, Oct. 1978.

21. J.F. Wakerly, "Transient Failures in Triple Modular Redundancy Systems with Sequential Modules", IEEE Trans. on Computers, Vol. C-24, pp. 570-573, May 1975.

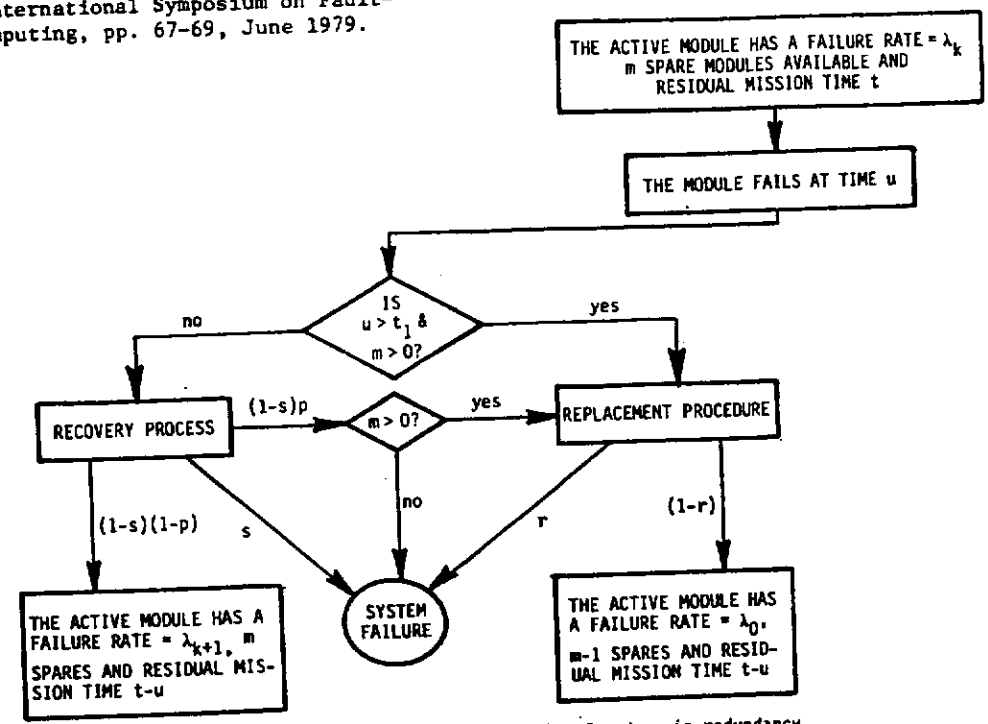


Figure 1. Module replacement policy for dynamic redundancy systems.

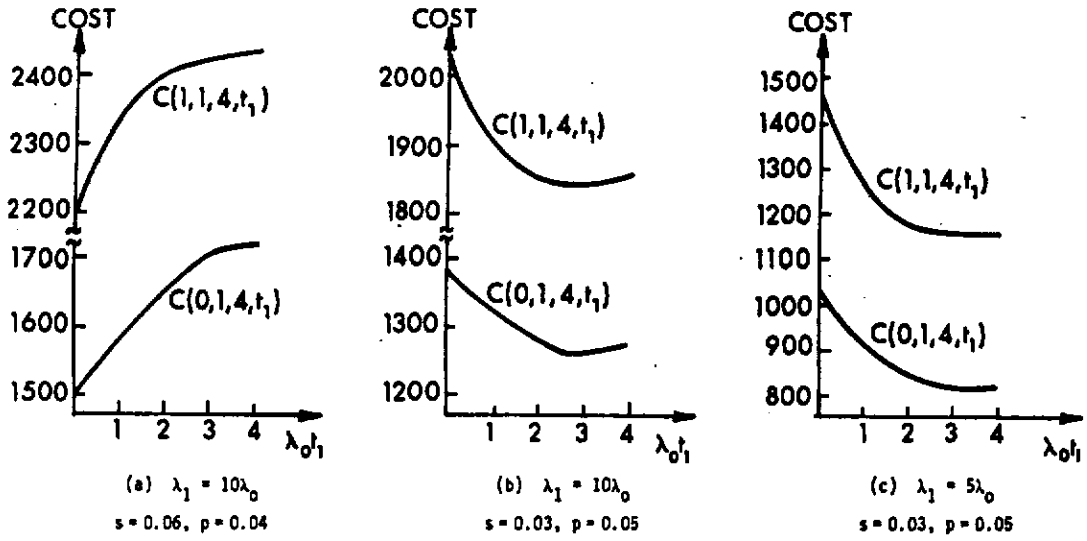


Figure 2. The dependence of $C(k,m,t,t_1)$ on $t_1$ for $C_3 = 1000C_1$, $C_2 = 2C_1$, and $r = 0.05$.

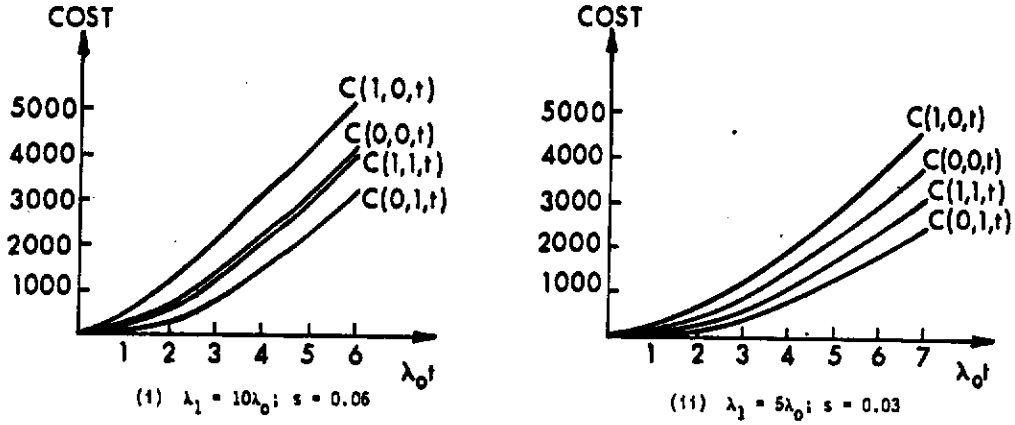


Figure 3. $C(k,m,t)$ as a function of the mission time t for $C_3 = 1000C_1$, $C_2 = 2C_1$, and $p = r = 0.05$.

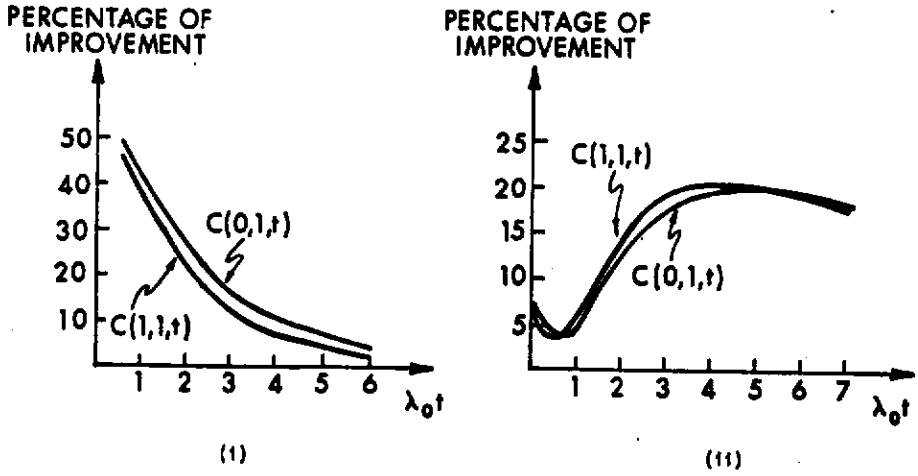


Figure 4. Percentage of maximal improvement in cost.