Available online at www.sciencedirect.com
SCIENCE DIRECT®
Reliability Engineering and System Safety 86 (2004) 75–81
RELIABILITY ENGINEERING & SYSTEM SAFETY
www.elsevier.com/locate/ress

Reliability and performance analysis for fault-tolerant programs consisting of versions with different characteristics

Gregory Levitin*

Planning, Development and Technology Division, Department of Reliability, The Israel Electric Corporation Ltd, P.O. Box 10, Haifa 31000, Israel

Received 22 June 2003; accepted 3 January 2004

Abstract

This paper presents a simple straightforward algorithm for evaluating reliability and expected execution time for software systems consisting of fault-tolerant components. The components are built from functionally equivalent but independently developed versions characterized by different reliability and performance. Both N-version programming (with parallel and sequential execution of the versions) and the recovery block scheme are considered within a universal model.

Keywords: Fault-tolerant programming; Performance; Reliability; Expected execution time

1. Introduction

Software failures are caused by errors made in various phases of program development. When the software reliability is of critical importance, special programming techniques are used in order to achieve its fault tolerance. Two of the best-known fault-tolerant software schemes are N-version programming (NVP) and recovery block scheme (RBS) [1]. Both schemes are based on the redundancy of software modules (functionally equivalent but independently developed) and the assumption that coincident failures of modules are rare.

NVP was proposed by Chen and Avizienis [2]. This approach presumes the execution of N functionally equivalent software modules (called versions) that receive the same input and send their outputs to a voter, which is aimed at determining the system output. The voter produces an output if at least M out of N outputs agree. Otherwise, the system fails. Usually majority voting is used in which N is odd and $M = (N + 1)/2$.

RBS was proposed by Randell [3]. This approach presumes consecutive execution of different versions. After execution of each version, its output is tested by an acceptance test block (ATB). If the ATB accepts the version output, the process is terminated and the version output is considered to be the output of the entire system. If the ATB does not accept the output, the next version is executed. If all N versions do not produce the accepted output, the system fails.

The fault-tolerant programming based on computational redundancy usually requires additional resources and results in performance penalties (particularly with regard to computation time), which constitutes a tradeoff between software performance and reliability. Estimating the effect of the fault-tolerant programming on system performance is especially important in safety critical real-time computer applications. This effect has been studied by Tai et al. [4] and by Goseva-Popstojanova and Grnarov [5,6]. While in Ref. [4] a basic realization of NVP ($N = 3, M = 2$) consisting of versions with identical fault probabilities and different execution times has been considered, in Refs. [5,6] NVP with arbitrary N has been studied in which both times to failure and execution times of different versions are identically distributed random variables.

In many cases, the information about version reliability and execution time is available from separate testing and/or reliability prediction models [7]. This information can be incorporated into a fault-tolerant program model in order to obtain a more precise evaluation of reliability and performance. The reliability model of NVP with versions having different reliability has been considered in Ref. [8]. However, in this study, the system performance evaluation problem has not been addressed and a general algorithm for evaluating NVP reliability for arbitrary N and M has not been suggested.

* Tel.: +972-4-818-3726; fax: +972-4-818-3790.
E-mail address: levitin@iec.co.il (G. Levitin).

0951-8320/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ress.2004.01.002

Nomenclature

C	number of components in the software system
M_c	number of identical outputs needed for component c to succeed
N_c	number of versions in component c
p_{ci}	probability that system component c produces correct output after execution of ith version
$\Pr(e)$	probability of event e
q_i	$\Pr(T = t_i)$
r_{ci}	reliability of ith version of component c
t_{ci}	time needed for system component c to produce the correct output after execution of ith version
t_i	ith realization of T
T	random execution time for the entire system
T_c	random execution time for system component c
T^*	upper bound for system execution time
τ_{AT}	acceptance test time
τ_{ci}	execution time of ith version of component c
τ_v	decision making time of the voter
X	number of different realizations of T
$1(x)$	function: $1(\text{TRUE}) = 1$, $1(\text{FALSE}) = 0$

Acronyms

ATB	acceptance test block
MGF	moment generating function
NVP	N-version programming
RBS	recovery block scheme

This paper presents an algorithm for evaluating the reliability and the performance of NVP and RBS with arbitrary N and M consisting of versions characterized by different reliability and execution time.

The models of different fault-tolerant programs and measures of their reliability and performance are presented in Section 2. A fast algorithm for evaluating the reliability and performance measures is presented in Section 3. Section 4 contains illustrative examples.

2. The models

According to the model presented in Ref. [8], the software system consists of C components. Each component performs a subtask and the sequential execution of the components performs a major task.

It is assumed that N_c functionally equivalent versions are available for each component c. Each version has an estimated reliability and execution time. Failures of versions for each component are statistically independent as well as the total failures of the different components.

In the simplest realization of NVP, execution of all versions begins simultaneously and after all N_c versions produce their outputs, the comparison is performed by the voter. If there are at least M_c identical outputs the component succeeds to perform the subtask, otherwise the component fails. The execution time of the component is equal to the execution time of the slowest version plus the time needed by the voter to make the decision (the latter time can usually be neglected).

Following the generalization of an idea suggested in Ref. [4], the versions' outputs could be compared each time the output of a new version becomes available (when the total number of completed versions is not less than M_c). If the comparison shows that the outputs of M_c different version coincide, the subtask execution is successfully terminated, otherwise the execution proceeds until all of the versions are executed. The component fails if after the execution of all N_c versions, the number of identical outputs is less than M_c. The entire component execution time is equal to the execution time of the version that has produced the M_cth correct output (Fig. 1A) plus the time needed by the voter to make the decision. It can be seen that the component execution time is

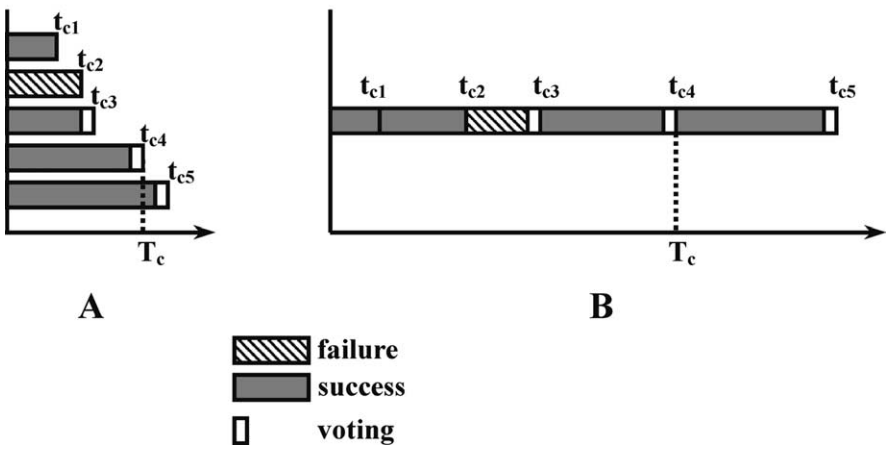

Fig. 1. Examples of NVP with parallel (A) and sequential (B) version execution for $N = 5$, $M = 3$.