This article has been accepted for publication in a future issue of a journal/proceedings, but has not been fully edited. Content may change prior to final publication. Citation information: Belli, F., Jedrzejowicz, P., Comparative Analysis of Concurrent Fault Tolerance Techniques for Real-Time Applications, Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE), pp.202-209

Comparative Analysis of Concurrent Fault Tolerance Techniques for Real-Time Applications

Fevzi Belli
University of Paderborn, GERMANY

Piotr Jedrzejowicz
Merchant Marine Academy Gdynia, POLAND

Summary

Consensus recovery block scheme [7] and concurrent recovery scheme [3] may become useful for real-time applications as they consider concurrency and time efficiency.

This paper compares the performance of both techniques in terms of the times they consume to achieve software fault tolerance. For this purpose a simple model will be used and analysed which simulates different real-time situations.

Keywords: Software fault tolerance, simulation, concurrency, real-time data processing

1. Introduction

In a recent paper Belli and Jedrzejowicz [3] reviewed and extended available techniques for achieving fault tolerance within computer programs. For each suggested technique a reliability model based on fault-tree analysis was suggested.

Reliability models show that the most promising approach towards constructing fault-tolerant programs, at least from the point of view of software systems reliability, includes consensus recovery block schemes introduced by Scott, Gault and McAllister [7], and concurrent recovery block schemes proposed by Belli and Jedrzejowicz [3]. Incidentally, both discussed approaches rely on using concurrent processing, although, the concurrency as such does not influence directly program reliability in either case. What it does influence is, of course, computation time. Just how much these times are affected and what would be the performance characteristics of both approaches remains an open question.

In this paper a comparative analysis of both - consensus recovery block scheme and concurrent recovery block scheme is attempted.

It has been asssumed that both techniques, because of their inherent concurrency and hence, an expected good performance in terms of computation times, may become useful when constructing real-time systems. Consequently, the simulation model used to compare both techniques has been designed to reflect fault-tolerant program behaviour within the real-time environment. Yet, the results of simulation have only a relative value and can be used to draw conclusions with respect to comparison between the two approaches only. These results are particularly not intended to be interpreted as a source of information on individual program performance.

The following sections cover simulation models formulation and implementation. Experiments that have been carried out, and their results are described. Conclusions referring to comparison between the analysed techniques and limitations of the simulation based on the analysis are drown.

2. Fault Tolerance Techniques for Concurrent Programs

2.1 Simple Recovery Block Scheme (RBS)

Assumptions [1,6]:

- Each program consists of n modules.

- Each module consists of a sequence of instructions (functional segment).
- There exists a testing and recovery segment checking against the result of computations at the end of module #1. If the acceptance test detects an erroneous output then the input state is recovered and module #2 activated. The procedure is repeated until acceptance, or the acceptance test judges the last module as incorrect (regardless of its actual correctness). The testing and recovery segment can, itself, be viewed as a module.

2.2 Consensus Recovery Block Scheme

This scheme, proposed by Scott, Gault, McAllister [7], requires the n-version of a program [2,4], a testing segment performing the acceptance check, and a decision algorithm with a voting procedure. The test procedure is performed as a sequence of stages.

Assumptions:

- Each version of a module representing the functional part of a program is mutually independent of all other versions.
- All paths (functional modules) can run concurrently
- Upon invocation of the Consensus RBS, all paths execute and submit their outputs to a decision algorithm. If at least 2 results agree on one output, that output is designated as correct. Otherwise the next stage is entered.
- At this stage the "best" path is examined by the acceptance test. If the output of that stage is accepted then it is treated as correct, else the next best version output is subject to testing. This process continues until an acceptable output is found, or the n outputs are exhausted.

2.3 Concurrent Recovery Block Scheme

Additional Assumptions (to Simple RBS) [3]

- Each program consists of n modules.
- Each modul consists of 2 segments (set of instructions): Functional and test. The test segment evaluates the performance (error free or failed) of the functional segment.

The test segment is subject to the same two kinds of failures as the Simple RBS.

All modules (paths) are run concurrently. Evaluation results are reviewed beginning with the possibly "best" result. The first accepted performance is considered as the ultimate and the next program is activated.

3. Model Formulation

3.1 The Problem

The problem at hand is to compare and analyse differences between the time-related performance characteristic of the alternative fault-tolerant programs. To arrive at a meaningful comparison it has been assumed that the processing takes place within a simple real-time system consisting of the single arrival channel, the queue and a single server (i.e. one of the two fault-tolerant programs itself).

The above mentioned system is shown in Fig. 1. Requests for computation arrive and eventually wait for processing in queue. They are processed in turn on the first come first served basis (see also [5]).

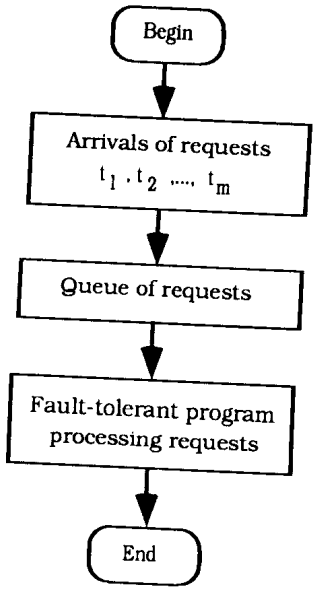


Figure 1: Simulated system

Arrivals are stochastic and can be modeled by any suitable distribution. Processing times are also stochastic but related to the internal structure and internal properties of the fault-tolerant programs. As we are unable, at this stage, to model time performance characteristics of the fault-tolerant program analytically, and do not want to treat it as a black-box, the only way to observe the behavior of the requests at the queue as well as the processing times is to build a simulation model and perform series of experiments.

In particular the simulation model at hand is used to assess relation and/or influence between various factors considered as the decision variables and various performance measures known as the observed variables.

Set of the decision variables includes:

- mean time between arrivals for requests for processing
- mean probabilities of failure of a program versions on a random input,
- number of multiple versions within a fault-tolerant program,
- measure of correlation among program versions failure within a fault-tolerant program,
- type of the fault-tolerant program (concurrent recovery block scheme or consensus recovery block scheme).

Set of the observed variables includes:

- statistics for the times that requests spent within the system (including waiting),
- statistics for the waiting times before the requests are actually processed,
- statistics for the processing times of requests,
- statistics for the systems reliability.

3.2 Main Assumptions

As usual, when building a simulation model several assumptions are required to justify necessary simplifications and generalizations. Majority of these refer to the underlying probability principles that govern the course of events within the modeled system. To model this course of events the following theoretical distributions are assumed to be justified:

- request arrivals are modeled by instances of the exponentially distributed random variable with mean time between arrivals being a decision variable (or experiment factor),
- processing times by a single program version are modeled by instances of the normally distributed random variable with known parameters (i.e. mean processing times and their variances),
- times of testing the results by a single testing segment are modeled by instances of the normally distributed random variable with known parameters (i.e. mean testing times and their variances),
- review times of the test results are proportional to the number of the results reviewed modified by the normally distributed random factor,
- review times by the voter (or decision algorithm) are proportional to the computational efforts involved modified by the normally distributed random factor,
- number of the faults within a multiversion fault-tolerant program is modeled by instances of the beta-binomial distribution with known parameters which can be transformed into mean failure probability of a program version on a random input and measure of correlation between versions failure.

3.3 Control of Events within Simulation Model

Figures 2 and 3 show two generalized procedures for events control within the proposed model. The first procedure refers to the system with concurrent recovery block scheme mechanism. The second covers consensus recovery block scheme version.

```
PROCEDURE concurrent_RBS_system;

BEGIN
  set values for decision variables;
  set value for the simulation time;
  set simulation clock to zero;
  generate sequence of requests arrivals;

  WHILE simulation time is not fully spent DO

    WHILE the queue of requests is not empty DO
      update statistics with respect to time in queue;
      consider the lowest numbered request;
      generate processing and testing times for each
        program version;
      find program version with the longest total
        processing and waiting time;
      advance simulation clock accordingly;
      generate number of failures;
      determine and note system failure or success;
      generate time for reviewing test results;
      advance simulation clock accordingly;
      update statistics with respect to times in processing;
    END

    consider the next arrival event;
    update simulation clock accordingly;
    calculate the remaining simulation time;
  END

  print simulation results;
END concurrent_RBS_system.
```

Figure 2: Generalized simulation procedure for event control for the concurrent RBS

```
PROCEDURE consensus_RBS_system;

BEGIN
  set values for decision variables:
  set value for the simulation time;
  set simulation clock to zero;
  generate sequence of requests arrivals;

  WHILE simulation time is not fully spent DO

   WHILE the queue or request is not empty DO
    update statistics with respect to times in queue;
    consider the lowest numbered request;
    generate processing times for each functional part
      within each program version;
    find program version with the longest processing
      time;
    advance simulation clock accordingly:
    generate number of failures;
    calculate review time of the results by decision
      algorithm;
    advance simulation clock accordingly;

    IF decision algorithm has not agreed upon result
     THEN
      generate testing times for each program version;
      find program version with the longest testing
        time;
      advance simulation clock accordingly;
      generate time for reviewing test results;
      advance simulation clock accordingly;
    END
    determine and note system failure or success;
    update statistics with respect to times in processing;
   END

   consider the next arrival event;
   update simulation clock accordingly;
   calculate the remaining simulation time;
  END

  print simulation results;

END consensus_RBS_system.
```

Figure 3: Generalized simulation procedure for event control for the consensus RBS

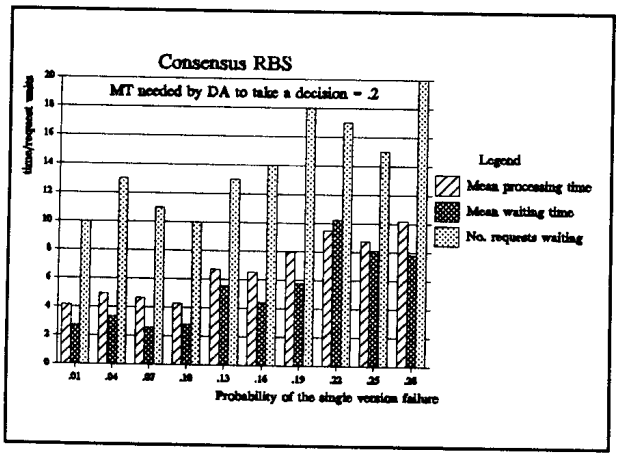
4. Experiment Plan and Results

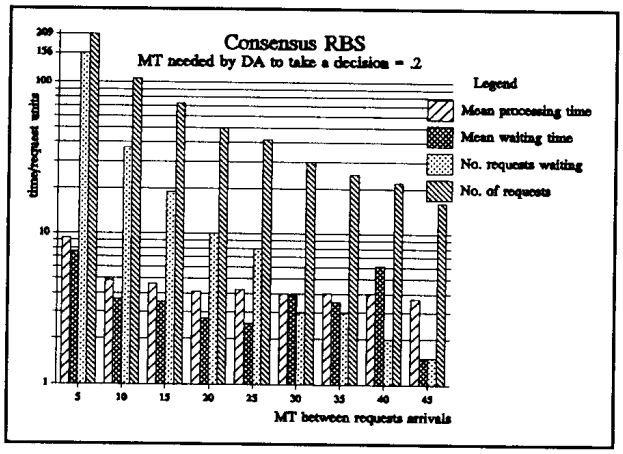
To compare some performance characteristics for systems based on concurrent recovery block scheme and on consensus recovery block scheme the following assumptions have been introduced:

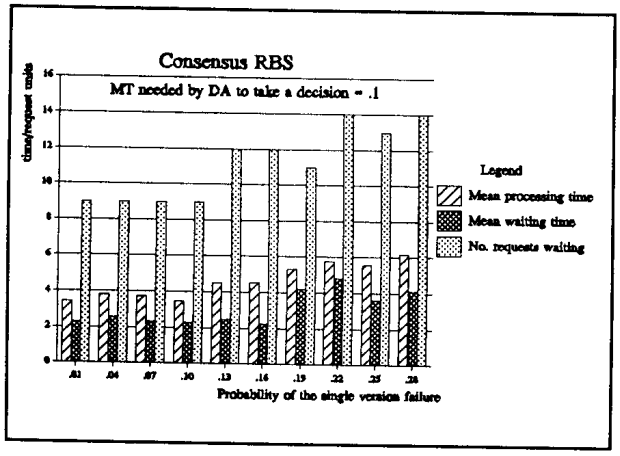
- mean processing times and their variances for a single functional version within a multiprogram system are identical in case of both systems;
- mean testing times and their variances required to evaluate the result of a single functional version computations are identical in case of both systems;
- mean times for the review of tests and their variances are identical in both systems;
- mean time needed by the decision algorithm to analyse one n-tuple of results generated by n-program versions is, alternatively equal to:
  a) mean review time required to evaluate the single test result
  b) half of the above
- variance of the above is identical with the variance for mean testing time.

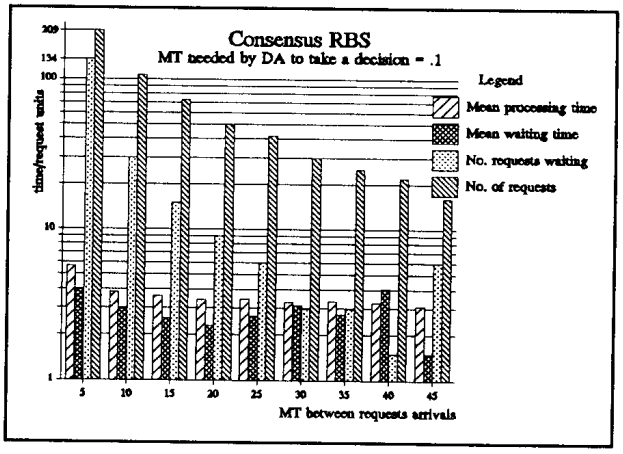
First experiment carried out (basing on the listed assumptions) aimed at investigating how the reliability of the single program version influences the overall system performance. The single analysed factor is probability of a single program version failure (p). It has been assumed that the number of program versions in each system is 6, mean time between requests arrivals is 20 and simulation period is 1000. Mean processing time in both cases is 2 with variance .5 and mean testing time 1 with variance .25. Mean review time per test result is .2 with variance .05. Mean time needed by the decision algorithm to analyse one n-tuple of results is .2 and .1, respecitvely. Its variance is .05. Correlation between versions failures is constant and equal .2. Experiment results are shown in Table 1.

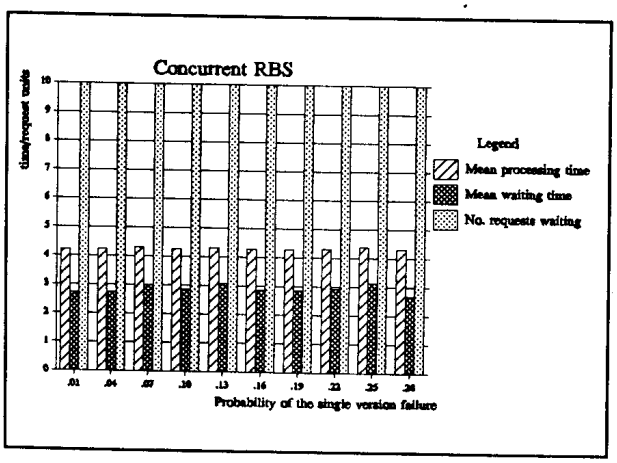
A second experiment aimed at investigating the influence of mean time between arrivals of requests for computation on system performance. All previous assumptions are valid except that probality of a single version failure is now constant and equal to .01. Table 2 shows the experiment results.

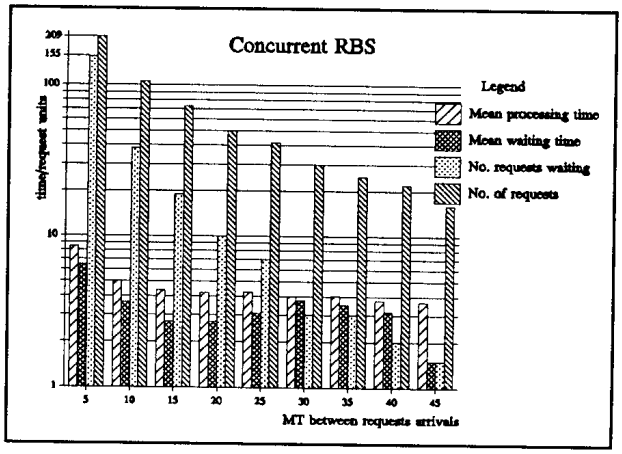








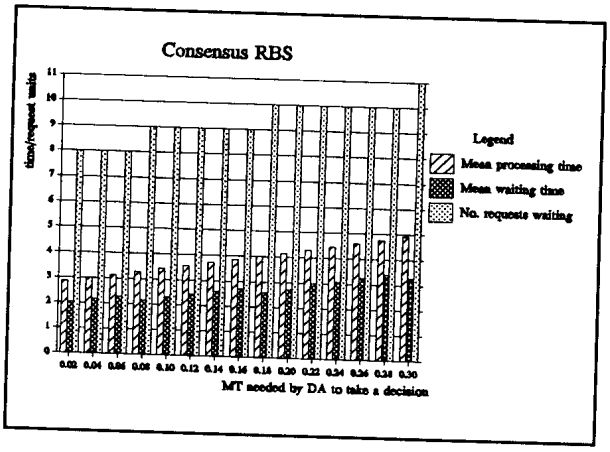




Tab. 1: Performance characteristics versus probability of the single version failure

Tab. 2: Performance characteristics versus mean time (MT) between arrivals of requests for computation

In Table 3 performance of the consensus RBS in relation to the time needed by the decision algorithm to analyse single n-tuple of results is shown. This performance has been the subject of the third experiment.



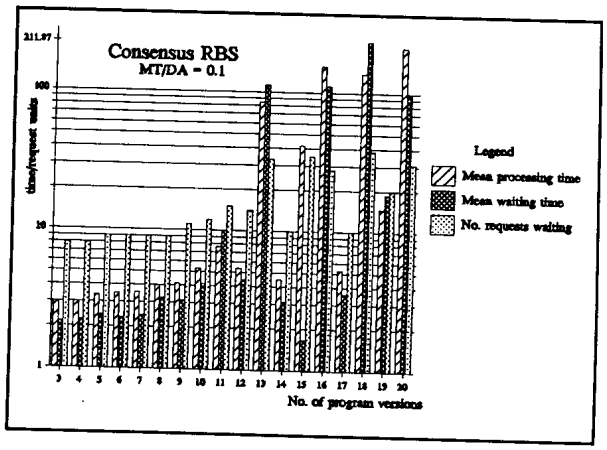
Tab. 3: Performance characteristics of the consensus RBS in relation to the time needed by the decision algorithm to analyse single n-tuple of results (mean results for 50 consecutive requests)

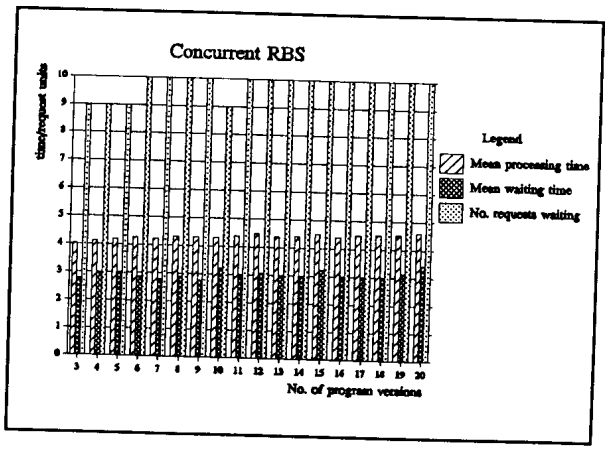
Finally, the fourth experiment aims at explaining how the number of redundant program versions may influence the system performance. The respective results are shown in Table 4.

5. Conclusions

Although the results obtained have preliminary character and no multi-factor experiments have been yet performed some conclusions as to the comparisons between time-related performance characteristics for concurrent and consensus recovery block scheme can be formulated.

Generally, as it has been expected, time performance characteristics of the concurrent RBS are better than these of the consensus RBS, especially when single program version reliability decreases and/or number of program versions increases. The reason behind this lays within the combinatorical nature of the decision algorithm (voter) of the consensus RBS. A number of n-tuples of results to be eventually analysed by the decision algorithm grows exponentially as the number of program versions increases. Also the increase of the number of failures of single version programs leads to the increase of the search space for the decision algorithm. This is clearly shown in Tab. 1. In case of the concurrent RBS mean times the requests spend in the system and mean waiting times for services are invariant to changes of reliability of single version programs.





Tab. 4: Performance characteristics in relation to the number of redundant program versions

Mean time between arrivals of requests is the factor clearly affecting time related performance characteristics of the both systems. The strength of this influence seems to be similar in both cases (see Tab. 2).

Any comparisons between both systems rely heavily on the relation between time required to test the results of computation by a single program version (in case of

the concurrent RBS) and time needed by the decision algorithm to analyse single n-tuple of results. To enable meaningful comparison, the experiment presented in Table 3 has been carried out. It shows the magnitude of influence of this particular factor on the consensus RBS performance.

Number or program versions influences positively the performance time characteristics for both approaches. As it is shown in Table 4 this influence in case of the concurrent RBS is substantially weaker. To arrive at more conclusive results further, more complex, experiments are needed. The value of the models would be also increased by using some real life data distributions describing the behaviour of the analysed systems.

Acknowledgments

This work was supported in part by the University of Paderborn.

REFERENCES

[1] T. Anderson, P. A. Lee, "Fault Tolerance: Principles and Practice", Prentice Hall, 1981

[2] A. Avizienis, "The n-version Approach to Fault Tolerant Software", IEEE Trans. Software Engineering, vol. SE-11, 1985 Dec., pp. 1491-1501

[3] Belli, F., Jedrzejowicz, P., "Fault-Tolerant Programs and Their Reliability", IEEE Transactions on Reliability, vol. 39, No. 2 (1990), pp. 184-192

[4] L. Chen, A. Avizienis, "N-version Programming: A Fault Tolerant Approach to Reliability of Software Operation", Proc. 8-th IFTCS, 1978, pp. 237-245

[5] F. W. Miller, "Predictive Deadline Multi-Processing", ACM Operating System Review, vol. 24, No. 4, 1990, Oct., pp. 52-62

[6] B. Randell, "System Structure for Software Fault Tolerance", IEEE Trans. Software Engineering, vol. SE-1, 1975 Feb., pp. 220-231

[7] K. Scott, J. W. Gault, D. F. McAllister, "Fault-Tolerant Software Reliability", IEEE Trans. Software Engineering, vol. SE-13, 1987 May, pp. 582-592

Appendix : Simulation Models

Variable List

n number of versions within multiversion software

t_i arrival time for the i-th request ($i = 1,..., m$)

T simulation period

CT calendar time

τ total time time for processing and testing a request

z'_j time of processing a request by the i-th program version ($j = 1,..., m$)

z'_j time of testing the results by the j-th testing segment ($j = 1,..., n$)

γ review time $\gamma = f(O(n))$

$X = \{x_i\}$ results of computations;

$$x_i = \begin{cases} 0 & \text{if the concurrent RBS failed for the i-th request} \\ 1 & \text{if the concurrent RBS succedded for the i-th request} \end{cases}$$

m number of arrivals during the simulation period

$U = \{u_i\}$ status of service;

$$u_i = \begin{cases} 0 & \text{if the i-th component is served} \\ 1 & \text{if the i-th request is not served} \end{cases}$$

$W = \{w_i\}$ waiting times of requests in cue

τ_o time of processing a request by the Consesus RBS

μ time of review of the results by DA (voter)

DA variable denoting the result of review by the DA.

$$DA = \begin{cases} 0 & \text{if the decision algorithm has not agreed on the result} \\ 1 & \text{otherwise} \end{cases}$$

τ_i time of testing the results by the Consesus RBS

The exact description of the experiments and their evaluations are included in our Technical Report 1990/6, which is available at:

Fevzi Belli, The University of Paderborn, FB 14, Pohlweg 47-49, D-4790 Paderborn, Germany

Concurrent RBS - Simulation Flow-Chart (Consensus RBS: analogously)