Performance Comparison of Different Software Fault Tolerance Methods

Md. Nasim Adnan, Mohammod Akbar Kabir, Lutful Karim, Nargis Khan

Abstract—A fault tolerance system is required for developing highly reliable computing systems that can function under adverse conditions, which is indispensable in safety critical applications. Fault tolerance is a major research issue in computing system designs because of the difficulty in producing error-free computing systems. This paper presents the recent development of software fault tolerance techniques and compares the performance of different software fault tolerant techniques and provides future research directions.

Keywords— Checkpoint, N version Programming, Recovery Blocks, Software Fault tolerance.

1 Introduction

FAULT tolerance is a major research area in computer system design because computers affect every aspect of modern life. Thus computing systems are required to operate without interruption for a long period of time. Fault tolerance makes systems capable of being operated in faulty conditions and protects against accidental or malicious destruction of information and generating erroneous output. It also ensures that confidential information cannot be divulged unintentionally. On the other hand, software is a key part of several critical applications, such as flight control systems and medical systems, as well as in real time systems. Therefore the researchers aim to develop fault tolerant software systems.

Despite the widespread use of software, it is extremely difficult to develop flawless software. In practice, at the end of the software testing phase, project managers assess software reliability (or quality) to fulfill a desired target. However, there is always the possibility that faults may be discovered later. This paper presents existing software techniques, identifies the limitations of these techniques and endeavours to provide solutions and further research directions in this area. Section 2 defines the important fault tolerant concept and terminology. In section 3, existing software fault tolerance techniques are presented. Section 4 discusses software fault detection techniques. In section 5 we identify certain limitations of existing fault detection and tolerance techniques, and present solutions and future research directions.

2 Terminology

Important terminology related to fault tolerance include [1]:

a) Fault

Fault is defined as an incorrect state of hardware or software resulting from physical defects, design flaws or operator error.

b) Fault Models

Depending on the system's behavior once a fault has occurred, the faults are characterized into different groups or classes.

c) Error

An error is part of a system state that may lead to a failure, or the manifestation of a fault.

d) Failure

When a system or a module is designed, its behavior is specified. When it is in service, we can observe its behavior. If the observed behavior differs from the specified behavior it is referred to as a failure. Failure is also the system level effects of an error.

e) Crash Failure

A process undergoes crash failure when it permanently ceases to execute its actions. This is an irreversible change, excluded from *napping* failures, where a process may play dead for a finite period of time before resuming operation. In *fail-stop* models, neighbors (processes) detect the faulty process, which crashes.

f) Omission Failure

Consider a transmitter process sending a sequence of messages to a receiver process. If the receiver does not receive some of the messages sent by the transmitter, an omission failure occurs.

g) Transient Failure

The agent inducing this failure may be temporarily active, but it can make a lasting effect on the global state. The failure can affect the global state in an arbi-

- *Assistant Professor, Department of Computer Science & Engineering, University of Liberal Arts, E-mail:nasim_1547@yahoo.co.uk.*
- *Assistant Professor, Department of Economics, University of Dhaka, E-mail: akbar_kabir03@yahoo.com.*
- *Research assistant to complete PhD from a University in Canada. E-mail: sumon_lk@yahoo.com.*
- *Lecturer, Department of Computer Science, Daffodil International University, Dhaka, Bangladesh, Email: nargis_ju@yahoo.com.*

Manuscript received on 30 July 2011 and accepted for publication on 27 October 2011.

trary manner.

h) Byzantine Failure

A process behaves arbitrarily when a Byzantine failure occurs. It represents the weakest of all failure models as it allows every conceivable form of erroneous behavior.

i) Software Failure

In some cases, the execution of a program suffers from the degeneration of the run-time system due to 'memory leaks', leading to a system crash. There may be problems with the adequacy of specifications, as occurred during the 'Y2K' problem. Many of the failures, such as crash, omission, transient, or Byzantine can be caused by software bugs.

j) Temporal Failure

Real time systems require actions to be completed within a specific time period. When this timestamp is not met, a temporal failure occurs.

k) Software Reliability

According to ANSI's definition, software reliability is defined as the probability of failure-free software operation for a specified period of time, in a specified environment.

3 SOFTWARE FAULT TOLERANCE TECHNIQUES

In this section we present a number of software fault tolerance techniques [1], [2]. Software fault tolerance is basically divided into two groups: single version and multi-version software techniques. Single version techniques are concerned with single software by adding several types of mechanisms during the design phase, with a goal to detect, contain, and handle errors. Multi-version fault tolerance techniques use multiple versions of the same software in a structured way to ensure that design faults in one version do not cause system failure. Different software fault tolerance techniques are discussed below.

3.1 Single Version Software Fault Tolerance Techniques

Single-version fault tolerance is based on the use of redundancy applied to a single version of a piece of software to detect and recover from faults. Among others, single-version software fault tolerance techniques include considerations of program structure and actions, error detection, exception handling, checkpoint and restart, process pairs, and data diversity.

3.1.1 Software Structure and Actions

The software architecture provides the basis for the implementation of fault tolerance. Various types of software structure and actions are available. Among them, the most popular techniques include: Modularizing, Partitioning, System Closure, and Temporal Structuring. The use of modularizing techniques to decompose a problem into manageable components is as important to the efficient application of fault tolerance as it is to system design. Partitioning is a technique for providing isolation between functionally independent modules. System closure is a fault tolerance principle stating that no action is permissible unless explicitly authorized. Temporal structuring of the activity between interactive structural modules is also important for fault tolerance.

3.3.2 Checkpoint and Restart

For single-version software, there are few recovery mechanisms. The most useful mechanism is the checkpoint and restart mechanism. A restart or backward error recovery (Figure 1) has the advantage of being independent of the damage caused by a fault, applicable to unanticipated faults, general enough to be used at multiple levels in a system, and conceptually simple. There are two types of restart recovery: static and dynamic. A static restart recovery is based on returning the module of software to a predetermined state. This can be a direct return to the initial reset state, or to one of a set of possible states. The selection is based on the operational situation at the moment the error detection occurred. Dynamic restart uses dynamically created checkpoints that are snapshots of the state at various points during the execution. Checkpoints can be created at fixed intervals or at particular points during the computation, determined by an optimization rule.

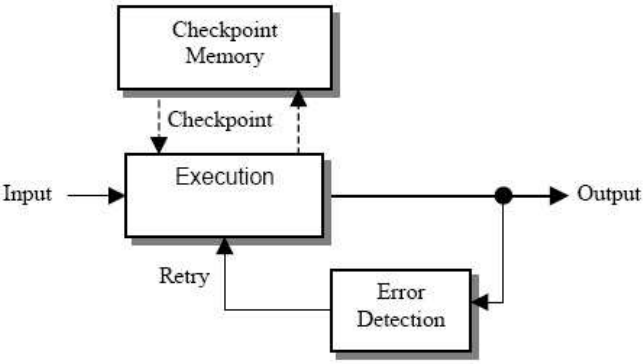

Figure 1: Logical Representation of Checkpoints and Restart

3.3.3 Process Pairs

A process pair uses two identical versions of software that run on separate processors (Figure 2). The recovery mechanism is checkpoint and restart. Two types of processors, namely primary and secondary processors, are used in this technique. Primary processors actively process the input and create output, as well as generating checkpoint information sent to the backup or secondary processors.

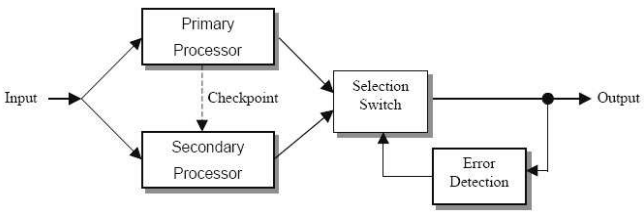

Figure 2: Logical Representation of Process Pairs

3.2 Multi-Version Software Fault Tolerance Techniques

Multi-version fault tolerance is based on two or more versions (or "variants") of a piece of software, executed either in serial or in parallel. The versions are used as alternatives (with a separate means of error detection) in pairs (to implement detection by replication checks) or in larger groups (to enable masking through voting).

3.2.1 Recovery Blocks

Fault tolerance techniques are alternate versions of a primary version of software used and the correct output (i.e. from all outputs of a primary version and alternative versions) is generated by a selection switch and an application dependent acceptance test. The recovery block technique increases the pressure on the specification to be specific enough to create multiple functional alternatives that are functionally the same.

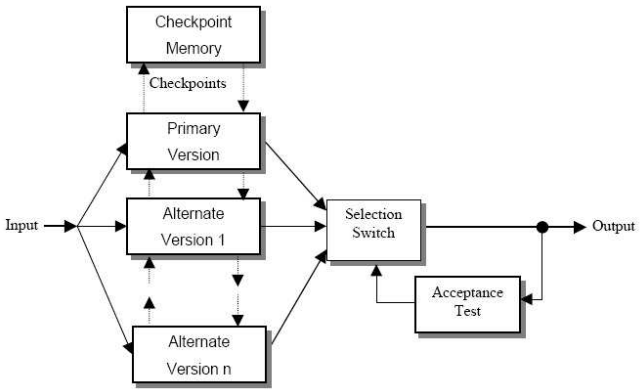

Figure 3: Recovery Block Model

3.2.2 N-Version Programming

N-Version programming [3] is a multi-version technique in which all versions are designed to satisfy the same basic requirements. The correctness of output is determined by comparing all outputs (Figure 4). The use of a generic decision algorithm (usually a voter) to select the correct output is a fundamental difference from the Recovery Blocks approach, which requires an application dependent acceptance test. This system can potentially overcome the design faults present in most software by relying on the design diversity concept.

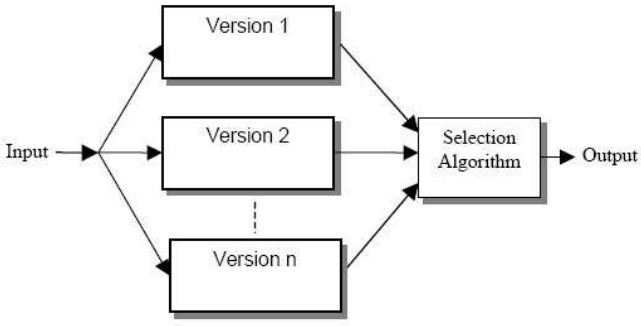

Figure 4: N Version Programming Model

3.2.3 N Self-Checking Programming

N Self-Checking programming uses multiple versions of software, combined with the structural variations of the Recovery Blocks and N-Version Programming. N Self-Checking programming using acceptance tests is shown in figure 5. Here, the versions and acceptance tests are developed independently from common requirements. The use of separate acceptance tests for each version is the main difference between the N Self-Checking model and the Recovery Blocks approach.

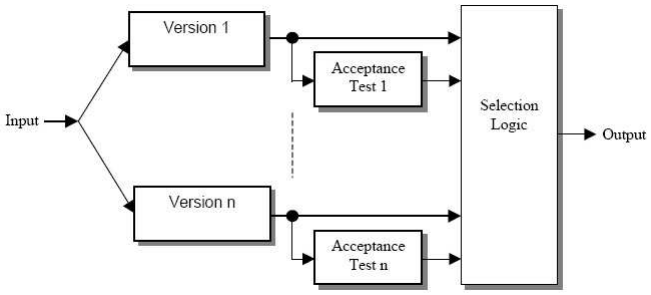

Figure 5: N Self-Checking Programming using Acceptance Tests

N self-checking programming using n-acceptance tests to compare the output for error detection is shown in figure 6. Like N-Version Programming, this model has the advantage of using an application independent decision algorithm to select a correct output. This variation of self-checking programming has a theoretical vulnerability of encountering situations where multiple pairs pass their comparisons, but with different outputs.

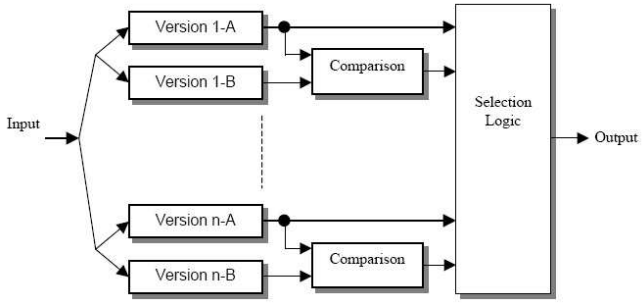

Figure 6: N Self Checking Programming using Comparison

3.2.4 Consensus Recovery Blocks

The Consensus Recovery Blocks (Figure 7) approach combines N-Version Programming and Recovery Blocks to improve the reliability achieved by using just one approach. The acceptance tests in the Recovery Blocks suffer from a lack of guidelines for development and a general tendency to design faults that are due to the inherent difficulty in creating effective tests. The use of voters, like in N-Version Programming, may not be appropriate in all situations, especially when multiple correct outputs are possible. In that case, a voter, for example, may result in failure when selecting an appropriate output. Consensus Recovery Blocks use a decision algorithm similar to N-Version Programming as a first layer of decision making. If this first layer finds a failure, a second layer using ac-

ceptance tests similar to those in the Recovery Blocks approach is invoked.

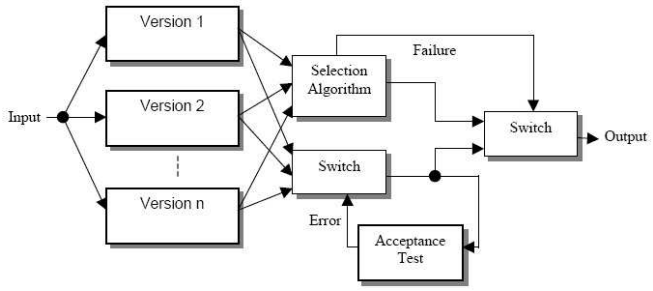

Figure 7: Consensus Recovery Blocks

4 SOFTWARE FAULTS DETECTION

In this paper [2], a number of techniques are proposed to detect and recover from transient faults. Transient faults (also known as soft errors), unlike manufacturing or design faults, do not occur consistently. To counter these faults, designers typically introduce redundant hardware such as RAID architecture, N-modular system, and error correcting code (ECC) to detect and recover from faults. However the hardware fault tolerant mechanisms are too expensive for many markets. On the other hand, software approaches to redundancy are attractive because they are essentially free of cost.

4.1 Error Detection by Duplication Instruction (EDDI)

EDDI [4] is a software-only fault detection system that operates by duplicating program instructions and using the redundant execution of programs to achieve fault tolerance. Program instructions are duplicated by the complier and are intertwined with the original program instructions. Each copy of the program, however, uses different registers and different memory locations so as to not interfere with one another. At certain synchronization points in the combined program code, the compiler makes sure that the original instructions inserts check instructions and their redundant copies agree on computed values. Since program correctness is defined by the output of a program, and if we assume memory-mapped I/O, then a program is executed correctly. Consequently, it is natural to use stored instructions as synchronization points for comparison. Unfortunately, it is insufficient to use the stored instructions as the only synchronization points, since misdirected branches may cause stored instructions to be skipped, incorrect stores to be executed, or incorrect values to feed a store. Therefore, branch instructions must also be synchronization points at which redundant values are compared.

4.2 Software Implemented Fault Tolerance (SWIFT)

SWIFT [4] is an efficient software-only, transient-fault detection technique. SWIFT efficiently manages redundancy by reclaiming unused instruction-level resources that are present during the execution of most programs. SWIFT makes several key refinements to EDDI and incorporates software only signature based control flow-checking scheme to achieve exceptional fault coverage. The major difference between EDDI and SWIFT is that while EDDI's SOR includes memory subsystems, SWIFT moves memory out of the SOR, as memory structures are already well-protected by hardware schemes such as parity and ECC, with or without scrubbing. SWIFT's performance greatly benefits from having only half of the memory subsystems.

5 ANALYSIS AND DRAWBACKS OF EXISTING SOFTWARE FAULT TOLERANCE TECHNIQUES

The methods discussed in the preceding sections are mostly used in critical and highly available systems. Fault tolerant techniques are highly reliable and available. However we found limitations in some software fault tolerance techniques. In the single version fault tolerant technique, reliability is achieved by sacrificing processing time [5]. On the other hand, in multi-version fault tolerance technique, the availability and reliability is achieved by using redundant components, which results in extra costs. Moreover, developing the multi-version software is more complex than for normal software [6],[7].

We note that software faults tend to be stated dependent and activated by particular input sequences. Although a component's reliability is an important quality measure for system level analysis, software reliability is hard to estimate and post-verification reliability estimation remains controversial. For some applications, software safety is more important than reliability and fault tolerance techniques used in those applications are aimed at preventing catastrophes. Single version software fault tolerance techniques include system structuring and closure, atomic actions, inline fault detection, exception handling, and checkpoint and restart. Process pairs exploit the state dependence characteristic of most software faults to allow uninterrupted delivery of services, despite the activation of faults. Similarly, data diversity aims to prevent the activation of design faults by multiple alternate input sequences. Multi-version techniques are based on the assumption that software built differently should fail differently, and thus, if one of the redundant versions fails, at least one of the others should provide an acceptable output. Recovery blocks, N-version programming [8], [9], N self-checking programming, consensus recovery blocks, and n / (n-1)-variant techniques were presented. However special consideration was given to multi-version software fault tolerance and output selection algorithms. Operating systems must be given special treatment when designing a fault tolerant software system because of the cost and complexity associated with their development, as well as their complexity for correct system functionality.

6 FUTURE WORK

Many techniques have been developed to achieve fault tolerance in software. Each technique must be tailored to the particular application. In this paper, some im-

portant fault tolerance techniques were reviewed and their characteristics identified. Future work may involve using this information to develop new and improved techniques. Attention ought to be paid to data diversity rather than design diversity.

7 Conclusion

This paper reviews existing software fault tolerance techniques and investigates the performance metrics of these techniques. We identified the characteristics of several software fault tolerance techniques. Finally, we analyzed the limitations of these techniques. The application of these techniques is relatively new to the area of fault tolerance. The differences between each technique provide some flexibility of application.

References

[1] W. Torres-Pomales, "Software Fault Tolerance: A Tutorial," *Langley Research Center, Hampton, Virginia,* October 2000.

[2] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan and David I., *Proceedings of the International Symposium on Code Generation and Optimization (CGO'05), IEEE,* August 2005.

[3] A. S. Vilkomir, L. David Parnas, B. V. Mendiratta,and E. Murphy, "Availability evaluation of hardware/software systems with several recovery procedures," *Proc. 29th Ann. International Computer Software and Applications Conference, IEEE*, 2005

[4] N. Oh. P.P. Shirvani, and E.J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Transactions on Reliability*, vol. 51, no. 1, pp.63-75, March 2002.

[5] C.Y Huang, C.T Lin, and C.C Sue, *Software Reliability Prediction and Analysis durng Operationa use*, 0-7803-8932, IEEE, 2005

[6] M. Reformat, E. Igbide, *Isolation of Software Defects: Extracting Knowledge with Confidence*, 0-7803-9093, IEEE, 2005

[7] X. Cai, M. R. Lyu, and M. A. Vouk, "An Experimental Evaluation on Reliability Features of N-Version Programming," *Proc. 16th IEEE International Symp. Software Reliability Engineering,* 2005.

[8] P. J. DENNING, "Fault Tolerant Operating Systems," *Computing Surveys*, vol. 8, no. 4, December 1976.

[9] A. Avizienis, *Toward Systematic Design of Fault- Tolerant Systems*, 0018-9162, IEEE, 1997.

Md. Nasim Adnan received M.Sc. in CSE from Bangladesh University of Engineering and Technology (BUET) and B.Sc. in CSE from Khulna University. He is currently working as an Assistant Professor in the department of Computer Science and Engineering, University of Liberal Arts Bangladesh (ULAB). He also served as a Deputy Director in ITOCD, Bangladesh Bank. His research interests include Software Engineering, Database Systems and E-commerce.

Mohammod Akbar Kabir received his M.Sc. and B.Sc. (Hons) in Computer Science from Dhaka University in 2000 and 1998 Repectively. Currently, he is working as an Assistant Professor in the dept. of Economics, University of Dhaka. He also served as an Assistant Director in ITOCD, Bangladesh Bank and as a Lecturer, Dept. of Computer Science, Dhaka City College, Dhaka. His research interests include VLSI design and E-commerce.

Lutful Karim has been a faculty member of Computer Science in several Internationally reputed universities in Bangladesh and abroad since 2000. He is currently working as a Research Assistant to complete PhD from a reputed university in Canada. He has authored several refereed conference publications and journals, and is a member of an organization committee and technical program committee in several international conferences. His research interests include Wireless Communications, Wireless Sensor Network, Fault Tolerance Computing Systems and E-commerce.

Nargis Khan worked as a faculty member in Daffodil International University (DIU) and University of Development Alternatives (UODA) in Bangladesh for about 4 years after completing a Bachelors degree in Computer Science and Engineering from Jahangirnagar University, Bangladesh in 2004. She has published several refereed conference and journal papers. Her research interests include Wireless Communications and Networks, Mobile Computing, Computer Networks and Security.