LITERATURE REVIEW REPORT

ABSTRACT

It is known that in today's life computer applications plays an essential role in modern society, including in medical devices, e-commerce, aircraft control system and many other applications. Failures of such systems may cause very severe consequences. Due to limitations of human capability, it is impossible to design a system which is free from errors. Although software developers try they best to ensure that the system is clean before it goes in to use, but the analysis show that many application failures in today's experience are caused by software errors which inherent from design faults during software development. Therefore, the need for software fault tolerance becomes increasingly important. To ensure the development of highly dependable systems different author have suggested various techniques for building software fault tolerance. This paper will review the three methods for software fault tolerance which are recovery blocks, N-version programming, and self-checking software and suggesting the best method among them.

KEYWORDS: software fault tolerance, N-version programming, recovery blocks, self-checking software.

METHODOLOGY: The methodology used to conduct this research is a Literature Review. Different survey and reviews have been done through some published papers that involve the research about designing a software fault tolerance.

1. INTRODUCTION

Fault tolerance is the ability of a system to cope with internal error and continue to perform its task correctly. The ultimate goal of fault tolerance is to boost the dependability of a system [2, 9]. Reliability, availability and safety are desirable features for any computing system, and are primary attributes of dependable system. All designers wish to develop a system which operates without a failure but the analysis shows that such a goal is not achievable because it is practically impossible to design a perfect system. No matter how carefully those systems are tested, debugged, and verified, design bugs will still become the causes of system failure [3] and it is reported that about 60-90% of current computer errors are caused by software faults [2].

Therefore if these failures are not handled well it may lead to a large consequence in our life, for example imagine if the airline control system fails what would happen to the life of the people in Also we have witness that many todays aTherefore, the idea toward designing software fault tolerance became very crucial and many researchers have suggested different approaches for designing software fault tolerance. According to [2,3,6,9], there are number of approaches for tolerating software fault such as recovery block, N-version programming (NVP), Consensus recovery block(CRB), Distributed Recovery Block (DRB), N-self checking version programming(NSCP), Roll-Forward Check Pointing Scheme (RFCS), Extended Distributed Recovery Block (EDRB).

2. SOFTWARE FAULT TOLERANCE TECHNIQUES

Software fault tolerance involves the use of techniques that enable the system to continue deliver a required service even if the design fault becomes active. Software fault tolerance techniques are designed to allow a system to tolerate any kind of design faults that remain in the system after its development. These techniques are implemented during the development of the software. Therefore, when a fault occurs, these techniques provide mechanisms to the software system to prevent the occurrence of the system failure. According to [9], software fault tolerance approaches are categorized in to two groups which are design diversity (multiple-version) and single design.

Design Diversity Software Fault Tolerance Approach

Design Diversity or multiple version software fault tolerance is based on the use of two or more versions of a piece of software which are executed either in sequence or in parallel. In this approach components of a system are built through independent designs but deliver the same service. The fundamental assumption of design diversity is that components build differently will fail differently. Thus, if one of the redundant versions fails, at least one of the others will provide an acceptable output. Examples of such techniques include recovery blocks, N-version programming and self-checking software.

Single-Design Software Fault Tolerance Approach

Single version techniques focus on adding mechanism into design, detecting and handling of errors caused by the design faults and therefore improving the Fault Tolerance of a single piece of software. [3] The goal of single version techniques is to determine the fault occurrence in a

system. Examples of single-version software fault tolerance techniques are include error detection, exception handling, checkpoint and rest, process pairs, and data diversity.

2.1 N-VERSION PROGRAMMING

N-version programming (NVP), also known as multi-version programming is a fault tolerance technique where multiples version of software system is developed, these versions are independently generated from the same initial specifications. With N-Version Programming, NVP, independent development teams use the same specification to generate multiple implementations. During development the design teams are kept separate and do not share their designs nor do they discuss the specification's meaning with each other. In this technique the design teams use different algorithms and different programming languages to produce multiple versions. These versions are either run sequentially in a single processor or parallel in a loosely coupled processor and then submits its answer to voter or decider or decision mechanism (DM) which determines the correct answer and returns the final result if it exist. The voter uses the result of majority to make decision of the final result. According to [5, 9], some of the generalized selection algorithms are Formalized majority voter, Generalized median voter, Formalized plurality voter and Weighted averaging techniques. Other voting techniques are based on Neural network and Genetic algorithm techniques. They are implemented such that their performance is related to the application and the particular characteristic of the software versions. It is hoped that by performing the N designs independently it will overcome the design faults present in most software, since the same mistakes will not be made in all the modules The voting module will be able to detect a fault because the same fault is not expected to occur in all the modules [5].

The General syntax of NVP:

```
run Version 1, Version 2, ..., Version n

if (Voter (Result 1, Result 2,...,Result n))

        return Result

else

        failure exception
```

FIGURE OF NVP

Consider the figure-1 which demonstrate how NVP work: The basic elements of the N-version programming approach are:

- ***The initial specification*** - this is the functionality that a software system is desired to provide
- ***N software versions*** - software modules which all are independently generated from the initial specification;
- ***A decision mechanism(voter)*** - a mechanism which take in the results from each version as input and make decisions on the final result.
- ***A supervisory program*** - this is a software structure used to drive the N versions and the decision mechanism.

2.2 RECOVERY BLOCKS

Recovery blocks [2,7,9,] is a software fault-tolerance technique which combines the basics of checkpoint and recover schemes. N versions of a software block are provided and a set of acceptance tests is used. One version of the block is designated as the primary and the rest are alternate versions. The Checkpoints are created before a version executes and are needed to recover the state of the software system after a version fails to produce a desired operation. The Acceptance Test (AT) need not be an output-only test and can be implemented by various embedded checks to increase the effectiveness of the error detection [9]. To perform a recovery block operation, the primary versions is executed and an acceptance test is run to determine whether the version has performed a valid operation. If the primary version fails to complete or fails the acceptance test, the system state is restored back to its current state before entry into the primary version, it does so by using checkpoint and an alternate version is performed. If the primary version passes the acceptance test, all alternates are ignored and if of all of alternate versions fails to pass the acceptance test, then the entire recovery block is considered to have failed, so the block in which it is embedded fails to complete [7].

The general syntax of RB is:

ensure Acceptance Test

by Primary Alternate

else by Alternate 2

else by Alternate 3

...

else by Alternate n

else failure exception

Consider the figure-2 which demonstrate how the RB techniques work

from figure-1, recovery blocks uses an Acceptance Test (AT) to accomplish fault tolerance.As the system operates, checks are made on the acceptability of the results generated by each software component. Should one of these checks fail, a spare component is switched on to take the place of the faulty component. The spare component is, of course, not merely a copy of the main component. Rather it is of independent design, so that there can be the possibility that it can cope with the circumstances that caused the main component to fail. Of course, recovery blocks satisfy this general description. The deadline mechanism [CAMP79] and dissimilar backup software are also included here because of their similarity with recovery blocks.[joshi]

2.3 SELF-CHECKING SOFTWARE

3. RESULTS AND DISCUSIONS

4. CONCLUSION AND FUTURE WORK

REFFERENCE

[1] Lee and R. K. Ayer, "Faults, Symptoms, and Software Fault Tolerance in the Tandem GUARDIAN90 Operating System", IEEE 1993, pp. 20-29.

[2] Dr. K. C. Joshi, “Approaches of Software Fault Tolerance Computing ", International Journal of Advanced Research in Computer Science and Software Engineering, Vol 5, Issue 9, September 2015.

[3] Jashan Deep and Rajiv Mahajan, “An Efficient Approach For Software Fault Tolerance In Parallel Computing”, International Journal of Computer Engineering & Science, March 2014.

[4] P. K. Lala and K. C. Yarlagadda, “On Self-checking software design”, IEEE, 1991.

[5] Bharathi V, “N-Version programming method of Software Fault Tolerance: A Critical Review”, National Conference On Nonlinear Systems & Dynamics, NCNSD-2003

[6] Anjushi Verma, Ankur Ghaartan and Tirthankar Gayen, “Review of Software Fault-Tolerance Methods for Reliability Enhancement of Real-Time Software Systems”, International Journal of Electrical and Computer Engineering (IJECE), Vol. 6, No. 3, June 2016, pp. 1031 – 1037.

[7] Sunita and Shekher Dahiya, “Review Paper on Recovery of Data during Software Fault”, International Journal of Innovative Research in Advanced Engineering (IJIRAE), Volume 1 Issue 2 (April 2014).

[8] NANCY G. LEVESON, STEPHEN S. CHA, JOHN C. KNIGHT AND TIMOTHY J. SHIMEALL, “The Use of Self Checks and Voting in Software Error Detection: An Empirical Study”, IEEE TRANSACTIONS OF SOFTWARE ENGINEERING. VOL 16. NO 4. APRIL I990.

[9] Goutam Kumar Saha, “Approaches to Software Based Fault Tolerance – A Review”, Computer Science Journal of Moldova, vol.13, no.3(39), 2005.

[10] Lance Fiondella and Panlop Zeephongsekul, “Recovery Block Fault Tolerance Considering Correlated Failures”, IEEE 2014.

[11] Laprie, J., C., and K., “Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures,” IEEE Computer, Vol. 23, No. 7, 1990, pp. 39-51.