An Approach to Build Software Based on Fault Tolerance Computing Using Uncertainty Factor

Mrityunjay Brahma

Department of Computer Science,
MIPS, MITS Rayagada, Odisha 765017, India

Abstract— **In this work, we have started with an overview on fault tolerance based system. In case of design diversity based software fault tolerance system, we observed that uncertainty remains an important factor. Keeping this factor, we have discussed about implementing Bayes' theorem and probabilistic mathematical model to handle the uncertainty factor. We assume that, once developed, the complete model will give us better efficiency. The rest of this paper deals with other types of fault tolerance systems and their approaches. This part is a kind of literature review, which includes, fault tolerant computing schemes that rely on the single-design as well as on the multiple-design. Further, in single-design, we have discussed about recovery block, N-version programming, N self-checking programming scheme. Lastly, focusing on multiple-design, we have discussed about software engineering aspects, error detection mechanisms and fault tolerance by fault injection. The paper ends with a general conclusion.**

Keywords— **Fault tolerance, Software fault tolerance, Bayes' Theorem, Uncertainty.**

I. Introduction

Most fault-tolerant computer systems are designed to be able to handle several possible failures, including hardware-related faults such as hard disk failures, input or output device failures, or other temporary or permanent failures; software bugs and errors; interface errors between the hardware and software, including driver failures; operator errors, such as erroneous keystrokes, bad command sequences, or installing unexpected software; and physical damage or other flaws introduced to the system from an outside source. Hardware fault-tolerance is the most common application of these systems, designed to prevent failures due to hardware components. Typically, components have multiple backups and are separated into smaller "segments" that act to contain a fault, and extra redundancy is built into all physical connectors, power supplies, fans, etc. There are special software and instrumentation packages designed to detect failures, such as fault masking, which is a way to ignore faults by seamlessly preparing a backup component to execute something as soon as the instruction is sent, using a sort of voting protocol where if the main and backups don't give the same results, the flawed output is ignored. Research into the kinds of tolerances needed for critical systems involves a large amount of interdisciplinary work. The more complex the system, the more carefully all possible interactions have to be considered and prepared for. Considering the importance of high-value systems in transport, public utilities and the military, the field of topics that touch on research is very wide: it can include such obvious subjects as software modeling and reliability, or hardware design, to arcane elements such as stochastic models, graph theory, formal or exclusionary logic, parallel processing, remote data transmission, and more. But without constant collaboration and data/instruction supply, no system can work. Uninterrupted information supply plays the most vital role in successful project management systems. For the case of project, several devices are attached together such as Servers, data storage facility, client machines, networking devices and so on, all supported by different software. These combinations of systems and software works round the clock in basic of 24x7x365 days [3],[5],[6].

Fault tolerance is a technique so that a system perform its function correctly even in the presence of internal faults. The purpose of fault tolerance is to increase the dependability of a system. A failure occurs when a system deviates from the specified behavior. This type of failure is called an error. Fault tolerance techniques are used to tolerate fault by redundancy [4],[7].

Software faults are commonly called "bugs". Software fault tolerance techniques are designed to allow a system to tolerate software faults that remain in the system after its development. It provides protection against errors in translating the requirements and algorithms into a programming language, but do not provide explicit protection against errors in specifying the requirements. These techniques have been used in the aerospace, nuclear power, healthcare, telecommunications and ground transportation industries, among others [4],[11].

II. Designing Approaches for Software based Fault Tolerance

Various software based fault tolerant approaches that are generally rely on design diversity (multiple version) as well as on single design. In the following sections, we have discussed about them in a bit elaborative way [4].

A. *Design Diversity Based Software Fault Tolerance*

Within the preview of software fault tolerance concepts, the design diversity based is also known as multiple version based software fault tolerance. Design diversity mainly act as a protection against uncertainty. The main goal of this technique is to build program versions that fail independently and with low probability of coincidental failures. Probability means chances of occurrence or predictability. The calculation becomes more challenging, when partial information related to the result is available. Again in some cases, even partial results are also not available. Bayes' theorem plays an important role in solving problems related to uncertainty. Here we have discussed about the scenario of handling uncertainty using Bayes' theorem and mathematical foundation of it[4],[8],[9].

B. *Uncertain or Incomplete Information related to Fault Tolerant system*

The task of decision making is intimately associated with every sphere of human life. In fact, the ability to make rational decisions is a unique characteristic. People have continuously devised means and ways to enlarge their abilities to cope with the growing complexity of their decision problems. Making decisions under uncertainty and imprecision is one of the most challenging problems of our age, which for a long time have been tackled by philosophers, logicians, and others. Recently, AI researchers have given new momentum and flavor to this idea. Expert systems, decision support systems, machine learning, inductive reasoning, pattern recognition, are areas where decision making under uncertainty is of primary importance. There are several mathematical models of uncertainty (e.g., Probabilistic functions, fuzzy set, rough set, theory of evidence and so on)[2],[8].

In ancient times, the logic was built as the science of reasoning on human knowledge, and the establishment of symbolic logic in the early years of the 20th century suggests the possibility of mechanical processing of this reasoning. Further, rapid progress of computer in these days enable is to realize this aim. In such a stream, the researches on knowledge have been one of the central topics of artificial intelligence. Originally, the logic has two aspects namely, the formal one (syntax) and the material one (semantics), and these two are strongly related with each other. For this reason, the problem of knowledge representation must be argued from the point of view of logic [1],[8].

Very often, while trying to discover knowledge from information systems, one may face the problem of missing values or uncertain values. In some cases partial information related to the results are available and in some other cases, even partial results are also not available. Several approaches to solve such type of the problem have been proposed. However, instead of discussing on extraction and discovery of knowledge, we have focused on prediction and chance of occurrence for an event in such type of problem. Thus our approach consists in removing rules with unknown values or replacing unknown values with the most common values or to make an approach for predicting the value using the concept of Probabilistic approach [8].

C. *Mathematical background*

In the 18th-century clergyman's theories on probability have become a major part of the mathematical foundations of application development. Search giant Google and Autonomy, a company that sells information retrieval tools, both employ Bayesian principles to provide likely (but technically never exact) results to data searches. Researchers are also using Bayesian models to determine correlations between specific symptoms and diseases, create personal robots, and develop artificially intelligent devices that "think" by doing what data and experience tell them to do. One of the more vocal Bayesian advocates is Microsoft. The company is employing ideas based on probability--or "probabilistic" principles--in its Notification Platform. The technology will be embedded in future Microsoft software and is intended to let computers and cell phones automatically filter messages, schedule meetings without their owners' help and derive strategies for getting in touch with other people. If successful, the technology will give rise to "context servers"--electronic butlers that will interpret people's daily habits and organize their lives under constantly shifting circumstances [8],[9],[10].

Bayesian research is used to make the best gambles on where I should flow with computation and bandwidth," said Eric Horvitz, senior researcher and group manager of the Adaptive Systems & Interaction Group at Microsoft Research. "I personally believe that probability is at the foundation of any intelligence in an uncertain world where you can't know everything" [1],[8].

D. *Bayes' Formula*

Let E and F be events. We may express *E* as $E = EF \cup EF^C$

For in order for an outcome to be in *E*, it must either be in both *E* and *F* or be in *E* but not in *F*.
As *EF* and EF^C are clearly mutually exclusive, we have

$$\begin{aligned} P(E) &= P(EF) + P(EF^C) \\ &= P(E \mid F)P(F) + P(E \mid F^C)P(F^C) \\ &= P(E \mid F)P(F) + P(E \mid F^C)[1 - P(F)] \end{aligned}$$

The above equation states that the probability of the event *E* is a weighted average of the conditional probability of *E* given that *F* has occurred and conditional probability of *E* given that *F* has not occurred – each conditional probability being given as much weight as the event on which it is conditioned has of occurring [9],[10].

E. Analysis of Bayes' Theory

Toward the end of the year, Intel will also come out with a toolkit for constructing Bayesian applications. As recently, improved mathematical models, faster computers and valid results from experiments have given new credibility to the school of thought.

Despite the esoteric symbols, the idea--roughly speaking--is simple: The likelihood that something will happen can be plausibly estimated by how often it occurred in the past. Researchers are applying the idea to everything from gene studies to filtering e-mail to noting down about the software crash and faults. Bayesian theory can roughly be described with one principle: To predict the future (about the occurance of fault), one must look at the past [1]. Bayes theorized that the probability of future events could be calculated by determining their earlier frequency. Will a flipped coin land heads up? Experimental data assigns it a value of 0.5. "Bayes said that essentially everything is uncertain, and one may have different distributions on probability," said Ron Howard, a professor in the Department of Management Science and Engineering at Stanford. He further described his points by an example, that instead of flipping a coin, if a researcher tossed a plastic pushpin and wanted to know what the chances were that it would land flat on its back with the pin pointing up, or, if it landed on its side or in what direction it would be pointing. In this case, shape, imperfections in the molding process, weight distribution and other factors, along with the greater variety of outcomes, would affect the results. We are working on this direction to relate this mathematical knowledge in finding uncertain fault and thus which may light us to propose a better fault tolerance system.

The appeal of the Bayesian technique is its deceptive simplicity. The predictions are based completely on data culled from reality--the more data obtained, the better it works. Another advantage is that Bayesian models are self-correcting, meaning that when data changes, so do the results [9]. Probabilistic thinking changes the way people interact with computers as well as how the new generation computers and machines will interact with each other. "The idea is that the computer seems more like an aid rather than a final device," said Peter Norvig, director of security quality at Google. According to him, "What you are looking for is some guidance, not a model answer." Search has benefited substantially from this shift. A few years ago, common use of so-called Boolean search engines required queries submitted in the "if, and, or but" grammar to find matching words. Now search engines employ complex algorithms to comb databases and produce likely matches. The same 'if', 'and' and 'but' will help to us to analyze and detect our proposed scenario under the preview of fault tolerance environment[8],[9].
However, analyzing through Bayes' model, we observed that few discount the importance of probability, debate on its uses lingers. Critics periodically assert that Bayesian models depend on inherently subjective data, leaving humans to judge whether an answer is correct. And probabilistic models do not completely account for the nuances in the human thought process.

III. Designing Approaches for Software based Fault Tolerance

In the following section, we have mainly focused on other software based fault tolerant approaches relying on design diversity (multiple version) and single design. This section is a kind of review of the existing literature in this direction [4].

A. The Recovery Block Scheme

With the continuation of this work, we again come back to literature of fault tolerance computing paradigm. The Recovery Block Scheme (RBS) technique reduces the software to crash by examining the checkpoints. It combined both the checkpoint and restart approach. Checkpoints are created before a version executes. Checkpoints are used to recover the state after a version fails to provide a valid operational starting point for the next version if an error is detected. In this case, the software executions can be sequential or parallel depending on the available processing capability and performance requirement. If all the alternates are tried unsuccessfully, the component must raise an exception to communicate to the rest of the system its failure to complete its function. The following figure (*Fig: 1.*) describes about the Recovery Block Scheme [7],[11].

B. The N-Version Programming Scheme

The N-Version programming Scheme is a multiple-version technique. The decision of output correctness is based on the comparison of all the outputs. Here task is executed by several processes or programs, this makes N-Version Programming Scheme as static technique [4]. Usually, generic decision algorithm (usually a voter) is used to select the correct output if one exits and it is the noted difference of this approach from the Recovery Blocks approach, which requires an application dependent acceptance test. The following figure (*Fig: 2.*) discusses about the N-Version programming Scheme [7],[11].

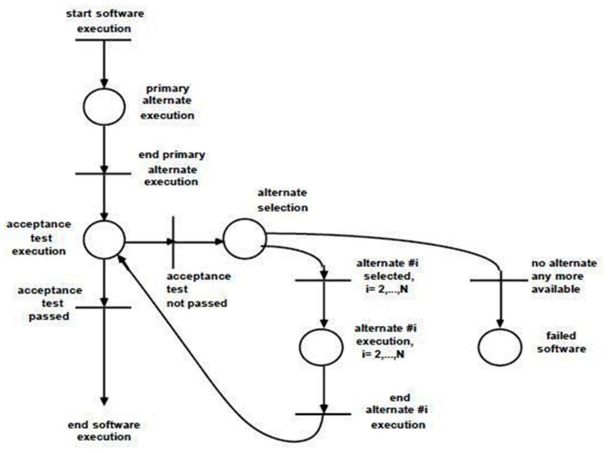
start software execution
primary alternate execution
end primary alternate execution
alternate selection
acceptance test execution
acceptance test not passed
alternate #i selected, i= 2,...,N
no alternate any more available
acceptance test passed
alternate #i execution, i= 2,...,N
failed software
end alternate #i execution
end software execution

Fig. 1 Recovery Block Scheme Process

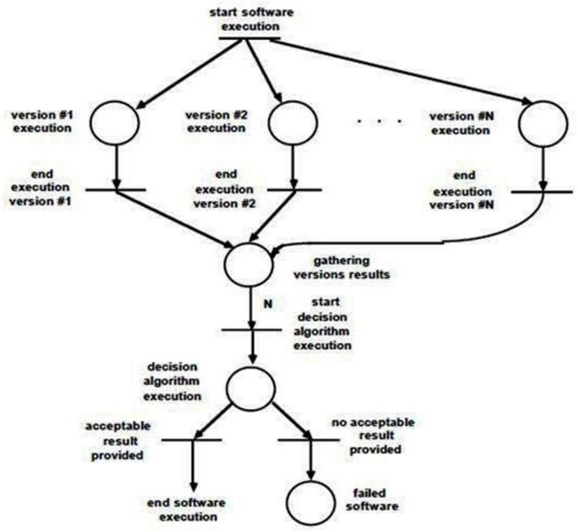

Fig. 2 N – Version Programming Scheme Process

C. *The N Self-Checking Programming Scheme*

The N Self-Checking Programming Scheme uses the multi software version combined with Recovery Blocks and N-Version Programming. N Self-Checking programming uses acceptance tests. Here the versions and the acceptance tests are developed independently from common requirements [4]. This use of separate acceptance tests for each version is the main difference of this N Self-Checking model from the Recovery Blocks approach. The Self-checking program uses comparison test for error detection. The figure (*Fig: 3.*) below, discuss about the process model [7],[11].

D. *Consensus Recovery Blocks Scheme*

The Consensus Recovery Blocks Scheme uses both N-Version Programming and Recovery Blocks to improve the reliability over that achievable by using just one of the approaches. The use of acceptance test in the Recovery Block Scheme, suffer from lack of guidelines for their development and a general proneness to design faults due to the inherent difficulty in creating effective tests [4]. The use of Voters in N-Version Programming does not support all the situations especially when there are multiple correct outputs. In that case a Voter would declare a failure in selecting an appropriate output. Consensus Recovery Blocks uses a decision algorithm similar to N-Version Programming as a first layer of decision. If this first layer declares a failure, a second layer using acceptance tests similar to those used in the Recovery Blocks approach is invoked.

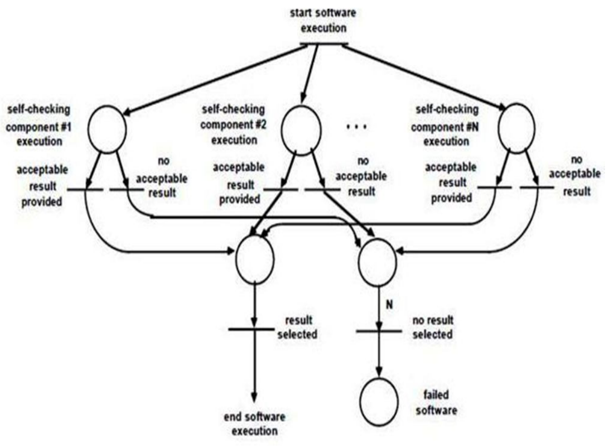
start software execution
self-checking component #1 execution
self-checking component #2 execution
. . .
self-checking component #N execution
acceptable result provided
no acceptable result
acceptable result provided
no acceptable result
acceptable result provided
no acceptable result
result selected
N
no result selected
end software execution
failed software

Fig. 3 N Self-Checking Programming Scheme Process

IV. Single-Design Software Fault Tolerance Approach:

Single-Design Software Fault Tolerance is based on redundancy. Redundancy is applied to the software to detect and recover from the faults [4].

A. *Software Engineering Aspects*

In Software Engineering Aspects, modularizing technique is used to decompose a large problem into small program for the efficient application of fault tolerance as it is to the design of a system. The modular decomposition of a design should consider built-in protections to keep aberrant component behavior in one module from propagating to other modules [4].

B. Error Detection Mechanisms

This fault tolerance technique in single version system uses two basic properties: self-protection and self-checking. The self-protection property means that a component must be able to protect itself from external contamination by detecting errors which is passed by other to it by other components. Self-checking means that a component must be able to detect internal errors and prevent the propagation of those errors to other components [4].

V. Assessment of Fault Tolerance by Fault Injection

Software fault injection is the process to test the software under any circumstances by injecting error. The main reason for using software fault injection is to ensure the quality of the software. Fault injection simulates software design faults by targeting the code [4]. In this situation, the injection considers the syntax of the software to modify it in various ways with the goal of replacing existing code with new code that is semantically different.

VI. Conclusion:

This paper deals with an overview on fault tolerance based system. In case of design diversity based software fault tolerance system, we observed that uncertainty remains an important factor. Keeping this factor, we have discussed about implementing Bayes' theorem and probabilistic mathematical model to handle the uncertainty factor. Focusing on this area, we have given a brief overview on handling uncertain and incomplete information related to fault tolerance system. We assume that, once fully developed, the complete model will give us better efficiency, at least for design diversity based software fault tolerance system. In rest of this paper, we have mentioned about other types of fault tolerance systems and their approaches. This part is a kind of literature review, which includes, fault tolerant computing schemes that rely on the single-design as well as on the multiple-design. Further, in single-design, we have discussed about recovery block, N-version programming, N self-checking programming scheme. Lastly, focusing on multiple-design, we have discussed about software engineering aspects, error detection mechanisms and fault tolerance by fault injection.

References

[1] Aman Gupta, Satyam Mandavalli, Vincent J. Mooney, Keck Voon Ling, Arindam Basu, Henry Johan and BudiantoTandianus, *Lower Probabilistic Floating Point Multiplier Design*, IEEE Computer Society Annual Symposium on VLSI, Chennai, India, July 4-6, 2011.

[2] B.K. Tripathy and Anirban Mitra, *On Rough Equalities and Rough Equivalences of sets*, Rough sets and current trends in computing, LNCS, Vol. 5306, 2008, pp. 92-102.

[3] Charlie Russel and Sharon Crawford, *Planning Fault Tolerance and Avoidance*, Chapter 7 - Microsoft Windows 2000 Server Administrator's Companion, Microsoft Press.

[4] G. K. Saha, *Approaches to Software Based Fault Tolerance – A Review*, Computer Science Journal of Moldova, Vol. 13, No. 3(39), 2005.

[5] *Internet site:* A fault-tolerant patent with a lot of basic information on specific ways to detect faults (http: //www.freepatentsonline.com/5099485.html)

[6] *Internet Site:* Wikipedia - the free encyclopaedia: Fault tolerance Computer architecture, Classes of computers, Fault-tolerant computer systems – ("http:/en.wikipedia.org/w/index.php?title=Fault_tolerant_computer_system&oldid=502750452")

[7] Jean-Claude Laprie, Jean Arlat, Christian Be´Ounes and Karama Kanoun, *Architectural Issues in Software Fault Tolerance*, Laas - Cnrs, France, Software Fault Tolerance, Edited By Lyu, 1995 John Wiley & Sons Ltd.

[8] Michael Itzenmacher, Eli Upfal, *Probability and computing: randomized algorithms and probabilistic analysis*, Cambridge University Press, 2005.

[9] Rejimon, T and Bhanja S, *Scalable probabilistic computing models using Bayesian networks*, South Florida University, Tampa, FL21 Midwest Symposium on Computers, February 2006.

[10] Sheldon Rose, *A first course in Probability*, 6th edition, Pearson Education.

[11] Zaipeng Xie, Hongyu Sun and Kewal Saluja, *A Survey of Software Fault Tolerance Techniques*, CiteSeerX (available at internet site: www.pld.ttu.ee).