A Literature Survey on Improving Fault Tolerance of Software Applications

S. Veera Pondy
Research Scholar
Department of Computer Science & Engineering
School of Information Technology, Madurai Kamaraj University, Madurai, India

Dr. S. Gavaskar
Assistant Professor
Department of Computer Science & Engineering
School of Computer Science & Engineering, Bharathiar University, Coimbatore, India

Dr. A. Sumithra
Associate Professor
Department of Computer Science & Engineering
School of Computer Science & Engineering, VSB College of Technical Campus, Coimbatore, India

Abstract

Adding fault tolerance to any software application is becoming an issue of great significance, especially as these applications support critical parts of our everyday life in the modern "Information Society". By adding this, the burden of ad hoc fault tolerance programming is removed from the application developer; while at the same time average fault tolerance support taken at operating system level is avoided. Fault-tolerance is achieved by applying a set of analysis and design techniques to create systems with dramatically improved dependability. As new technologies are developed and new applications arise, new fault-tolerance approaches are also needed. In the early days of fault-tolerant computing, it was possible to craft specific hardware and software solutions from the ground up, but now chips contain complex, highly-integrated functions, and hardware and software must be crafted to meet a variety of standards to be economically viable. Thus a great deal of current research focuses on implementing fault tolerance using COTS (Commercial-Off-The-Shelf) technology. In this paper we present a survey on fault tolerance provided in variety of ways.

Keywords: Software fault tolerance, fault-tolerant system, hardware faults, fault recovery, redundant

---

I. INTRODUCTION

Fault-tolerant computing is the art and science of building computing systems that continue to operate satisfactorily in the presence of faults. A fault-tolerant system may be able to tolerate one or more fault-types including -- i) transient, intermittent or permanent hardware faults, ii) software and hardware design errors, iii) operator errors, or iv) externally induced upsets or physical damage. An extensive methodology has been developed in this field over the past thirty years, and a number of fault-tolerant machines have been developed -- most dealing with random hardware faults, while a smaller number deal with software, design and operator faults to varying degrees. A large amount of supporting research has been reported. Fault tolerance and dependable systems research covers a wide spectrum of applications ranging across embedded real-time systems, commercial transaction systems, transportation systems, and military/space systems -- to name a few [1]-[3]. The supporting research includes system architecture, design techniques, coding theory, testing, validation, proof of correctness, modeling, software reliability, operating systems, parallel processing, and real-time processing. These areas often involve widely diverse core expertise ranging from formal logic, mathematics of stochastic modeling, graph theory, hardware design and software engineering.

A. Hardware Fault-Tolerance:

The majority of fault-tolerant designs have been directed toward building computers that automatically recover from random faults occurring in hardware components. The techniques employed to do this generally involve partitioning a computing system into modules that act as fault-containment regions. Each module is backed up with protective redundancy so that, if the module fails, others can assume its function. Special mechanisms are added to detect errors and implement recovery. [4][5]. two general approaches to hardware fault recovery have been used: 1) fault masking, and 2) dynamic recovery.

Fault masking is a structural redundancy technique that completely masks faults within a set of redundant modules. A number of identical modules execute the same functions, and their outputs are voted to remove errors created by a faulty module. Triple modular redundancy (TMR) is a commonly used form of fault masking in which the circuitry is triplicated and voted. The voting circuitry can also be triplicated so that individual voter failures can also be corrected by the voting process [6][7]. A TMR system fails whenever two modules in a redundant triplet create errors so that the vote is no longer valid. Hybrid redundancy is an extension of TMR in which the triplicated modules are backed up with additional spares, which are used to replace faulty modules -- allowing more faults to be tolerated. Voted systems require more than three times as much hardware as nonredundant

systems, but they have the advantage that computations can continue without interruption when a fault occurs, allowing existing operating systems to be used.

Dynamic recovery is required when only one copy of a computation is running at a time (or in some cases two unchecked copies), and it involves automated self-repair. As in fault masking, the computing system is partitioned into modules backed up by spares as protective redundancy. In the case of dynamic recovery however, special mechanisms are required to detect faults in the modules, switch out a faulty module, switch in a spare, and instigate those software actions (rollback, initialization, retry, and restart) necessary to restore and continue the computation. In single computers special hardware is required along with software to do this, while in multicomputer the function is often managed by the other processors. Dynamic recovery is generally more hardware-efficient than voted systems, and it is therefore the approach of choice in resource-constrained (e.g., low-power) systems, and especially in high performance scalable systems in which the amount of hardware resources devoted to active computing must be maximized. Its disadvantage is that computational delays occur during fault recovery, fault coverage is often lower, and specialized operating systems may be required.

B. Software Fault-Tolerance:

Efforts to attain software that can tolerate software design faults (programming errors) have made use of static and dynamic redundancy approaches similar to those used for hardware faults. One such approach, N-version programming, uses static redundancy in the form of independently written programs (versions) that perform the same functions, and their outputs are voted at special checkpoints. Here, of course, the data being voted may not be exactly the same, and a criterion must be used to identify and reject faulty versions and to determine a consistent value (through inexact voting) that all good versions can use. An alternative dynamic approach is based on the concept of recovery blocks. Programs are partitioned into blocks and acceptance tests are executed after each block. If an acceptance test fails, a redundant code block is executed. An approach called design diversity combines hardware and software fault-tolerance by implementing a fault-tolerant computer system using different hardware and software in redundant channels [8]-[11]. Each channel is designed to provide the same function, and a method is provided to identify if one channel deviates unacceptably from the others. The goal is to tolerate both hardware and software design faults. This is a very expensive technique, but it is used in very critical aircraft control applications.

II. HISTORY

The SAPO computer built in Prague, Czechoslovakia was probably the first fault-tolerant computer. It was built in 1950–1954 under the supervision of A. Svoboda, using relays and a magnetic drum memory. The processor used triplication and voting (TMR), and the memory implemented error detection with automatic retries when an error was detected. A second machine developed by the same group (EPOS) also contained comprehensive fault-tolerance features. The fault-tolerant features of these machines were motivated by the local unavailability of reliable components and a high probability of reprisals by the ruling authorities should the machine fail. Over the past 30 years, a number of fault-tolerant computers have been developed that fall into three general types: 1) long-life, unmaintainable computers, 2) ultradependable, Real-time computers, and 3) high-availability computers.

A. Long-Life, Unmaintained Computers:

Applications such as spacecraft require computers to operate for long periods of time without external repair. Typical requirements are a probability of 95% that the computer will operate correctly for 5–10 years. Machines of this type must use hardware in a very efficient fashion, and they are typically constrained to low power, weight, and volume. Therefore, it is not surprising that NASA was an early sponsor of fault-tolerant computing. In the 1960s, the first fault-tolerant machine to be developed and flown was the on-board computer for the Orbiting Astronomical Observatory (OAO), which used fault masking at the component (transistor) level. The JPL Self-Testing-and-Repairing (STAR) computer was the next fault-tolerant computer, developed by NASA in the late 1960s for a 10-year mission to the outer planets. The STAR computer, designed under the leadership of A. Avizienis was the first computer to employ dynamic recovery throughout its design. Various modules of the computer were instrumented to detect internal faults and signal fault conditions to a special test and repair processor that effected reconfiguration and recovery. An experimental version of the STAR was implemented in the laboratory and its fault tolerance properties were verified by experimental testing. Perhaps the most successful long-life space application has been the JPL-Voyager computers that have now operated in space for 20 years. This system used dynamic redundancy in which pairs of redundant computers checked each-other by exchanging messages, and if a computer failed, it's partner could take over the computations. This type of design has been used on several subsequent spacecraft.

B. Ultra dependable Real-Time Computers:

These are computers for which an error or delay can prove to be catastrophic. They are designed for applications such as control of aircraft, mass transportation systems, and nuclear power plants. The applications justify massive investments in redundant hardware, software, and testing. One of the first operational machines of this type was the Saturn V guidance computer, developed in the 1960s. It contained a TMR processor and duplicated memories (each using internal error detection)[1]. Processor errors were masked by voting, and a memory error was circumvented by reading from the other memory. The next

machine of this type was the Space Shuttle computer. It was a rather ad-hoc design that used four computers that executed the same programs and were voted. A fifth, non-redundant computer was included with different programs in case a software error was encountered.

During the 1970s, two influential fault-tolerant machines were developed by NASA for fuel-efficient aircraft that require continuous computer control in flight. They were designed to meet the most stringent reliability requirements of any computer to that time. Both machines employed hybrid redundancy. The first, designated Software Implemented Fault Tolerance (SIFT), was developed by SRI International. It used off-the-shelf computers and achieved voting and reconfiguration primarily through software. The second machine, the Fault-Tolerant Multiprocessor (FTMP), developed by the C. S. Draper Laboratory, used specialized hardware to effect error and fault recovery. A commercial company, August Systems, was a spin-off from the SIFT program. It has developed a TMR system intended for process control applications. The FTMP has evolved into the Fault-Tolerant Processor (FTP), used by Draper in several applications and the Fault-Tolerant Parallel processor (FTPP) -- a parallel processor that allows processes to run in a single machine or in duplex, triplex or quadrupled groups of processors. This highly innovative design is fully Byzantine resilient and allows multiple groups of redundant processors to be interconnected to form scalable systems [12]-[15].

The new generation of fly-by-wire aircraft exhibits a very high degree of fault-tolerance in their real-time flight control computers. For example the Airbus Airliners use redundant channels with different processors and diverse software to protect against design errors as well as hardware faults. Other areas where fault-tolerance is being used include control of public transportation systems and the distributed computer systems now being incorporated in automobiles.

C. *High-Availability Computers:*

Many applications require very high availability but can tolerate an occasional error or very short delays (on the order of a few seconds), while error recovery is taking place. Hardware designs for these systems are often considerably less expensive than those used for ultra-dependable real-time computers. Computers of this type often use duplex designs. Example applications are telephone switching and transaction processing. The most widely used fault-tolerant computer systems developed during the 1960s were in electronic switching systems (ESS), that are used in telephone switching offices throughout the country. The first of these AT&T machines, No. 1 ESS, had a goal of no more than two hours downtime in 40 years. The computers are duplicated, to detect errors, with some dedicated hardware and extensive software used to identify faults and effect replacement. These machines have since evolved over several generations to No. 5 ESS which uses a distributed system controlled by the 3B20D fault tolerant computer. The largest commercial success in fault-tolerant computing has been in the area of transaction processing for banks, airline reservations, etc.

Tandem Computers, Inc. was the first major producer and is the current leader in this market. The design approach is a distributed system using a sophisticated form of duplication. For each running process, there is a backup process running on a different computer. The primary process is responsible for check pointing its state to duplex disks. If it should fail, the backup process can restart from the last checkpoint. Stratus Computer has become another major producer of fault-tolerant machines for high-availability applications. Their approach uses duplex self-checking computers where each computer of a duplex pair is itself internally duplicated and compared to provide high-coverage concurrent error detection. The duplex pair of self-checking computers is run synchronously so that if one fails, the other can continue the computations without delay.

Finally, the venerable IBM mainframe series, which evolved from S360, has always used extensive fault-tolerance techniques of internal checking, instruction retries and automatic switching of redundant units to provide very high availability. The newest CMOS-VLSI version, G4, uses coding on registers and on-chip duplication for error detection, and it contains redundant processors, memories, I/O modules and power supplies to recover from hardware faults -- providing very high levels of dependability. The server market represents a new and rapidly growing market for fault-tolerant machines driven by the growth of the Internet and local networks and their needs for uninterrupted service. Many major server manufacturers offer systems that contain redundant processors, disks and power supplies, and automatically switch to backups if a failure is detected. Examples are SUN's ft-SPARC and the HP/Stratus Continuum 400.Other vendors are working on fault-tolerant cluster technology, where other machines in a network can take over the tasks of a failed machine. An example is the Microsoft MSCS technology. Information on fault-tolerant servers can readily be found in the various manufacturers' web pages.

III. VALIDATION OF FAULT-TOLERANCE

One of the most difficult tasks in the design of a fault-tolerant machine is to verify that it will meet its reliability requirements. This requires creating a number of models. The first model is of the error/fault environment that is expected. Other models specify the structure and behavior of the design. It is then necessary to determine how well the fault tolerance mechanisms work by analytic studies and fault simulations. The results, in the form of error rates, fault-rates, latencies, and coverages, are used in reliability prediction models. A number of probabilistic models have been developed using Markov and semi-Markov processes to predict the reliability of fault-tolerant machines as a function of time. These models have been implemented in several computer-aided design tools. Some of the better known tools are:

- HARP—Hybrid Automated Reliability Predictor (Duke)
- SAVE—System Availability Estimator (IBM)

- SHARPE—Symbolic Hierarchical Automated Reliability and Performance Evaluator
- (Duke)
- UltraSAN -- (University of Illinois, UIUC)
- DEPEND -- (UIUC)
- SURF-2 -- Laboratoire D'analyse Et D'architecture Des Systemes (LAAS)

Recently there has been a great deal of research in experimental testing by fault-insertion to aid in assessing the reliability of dependable systems. Among the fault-injection tools that have been developed to evaluate fault tolerant systems are: i) FTAPE (UIUC), ii) Ballista (CMU), and iii) MEFISTO (LAAS)[6][17].

IV. LITERATURE REVIEW

Amoon etal[18], in their Research **A** Fault Tolerant Scheduling System Based on Check pointing for Computational Grids says Job check pointing is one of the most common utilized techniques for providing fault tolerance in computational grids. The efficiency of check pointing depends on the choice of the checkpoint interval. Inappropriate check pointing interval can delay job execution. In this paper, a fault-tolerant job scheduling system based on check pointing technique is presented and evaluated. When scheduling a job, the system uses both average failure time and failure rate of grid resources combined with resources response time to generate scheduling decisions. The system uses the failure rate of the assigned resources to calculate the checkpoint interval for each job. Extensive simulation experiments are conducted to quantify the performance of the proposed system.

In E-science talks Anderson et al [19] says that today personal computers are powerful but, most of the time, a large proportion of their computational power is left unused. A desktop grid takes this unused capacity, no matter what its location, and puts it to work solving scientific problems. With over 1 billion desktop computers in use, desktop grids can offer a low cost, readily available computing resource for scientists while allowing citizens across the world to contribute to scientific research. Together with grids and supercomputers, desktop grids can be a useful complement to the e-Infrastructure landscape.

Azeez and Haque [20] in their research paper Resource Management in Grid Computing says as technology advances and the popularity and dependence on internet increases, advanced method of finding faster and cheaper solutions to computational problems are sought. It force lead to the development of a concept known as grid computing, which is a type of distributed computing, heterogeneous in nature because it makes an aggregated use of resources distributed over a large geographical area to solve problems usually complex ones on a larger scale. This paper provides a brief overview on grid computing and its resource management processes, important factors considered in resource management, comparison of different resource management processes and future outlook of grid computing and resource management.

Azeez et al [21], in their paper grid computing with alchemi says middlewares allow submission of requests to execute a computation (called a Job) to the Grid, such that it can be run anywhere on the network. Therefore, grid Middlewares serve as an intermediary layer that allow a reliable and homogeneous access to resources managed locally with different syntax and access techniques. Within the context of availability of various Middlewares for Grid implementation with different features, this paper therefore focuses on various features that are peculiar to Alchemi by taking into consideration its Architecture, the Operating systems, software demand and limitation that are inherent from its usage.

In their paper on fault tolerance of resources in computational grid Das and Sarkar [22], says various heterogeneous resources of different administrative domain are virtually distributed through different network in computational grids. Thus any type of failure can occur at any point of time and job running in grid environment might fail. Hence fault tolerance is an important and challenging issue in grid computing as the dependability of individual grid resources may not be guaranteed. In order to make computational grids more effective and reliable fault tolerant system is necessary. The objective of their paper is to review different existing fault tolerance techniques applicable in grid computing. The paper presents state of the art of various fault tolerance technique and comparative study of the existing algorithms.

Dhir et.al [23] says in nimble@itccnogrid vs alchemi .net development of a novel toolkit namely Nimble@ITCEcnoGrid for calculating the pi() value up to 120 decimal places after decimal. Alchemi.NET is the other popular toolkit based on Windows Platform for doing the same purpose. There is a very interesting comparison between Nimble@ITCEcnoGrid toolkit and Alchemi toolkit as both runs on Windows Operating System.

Ebeneze and Baskaran [24] in their paper Fault Tolerant most Fitting Resource Scheduling Algorithm for Computational Grid proposes a replication technique to improve the fault tolerance of the fittest resource scheduling algorithm .Scheduling the task to the appropriate resource is a vital requirement in computational Grid. The fittest resource scheduling algorithm searches for the appropriate resource based on the job requirements, in contrary to the general scheduling algorithms where jobs are scheduled to the resources with best performance factor. They proposed a method to improve the fault tolerance of the fittest resource scheduling algorithm, by scheduling the job in coordination with job replication when the resource has low reliability

Minhas et.al [25] in their paper A novel cost-based framework for communication in computational grid using Anycast Routing says Computational grid can perform the computationally extensive jobs by utilizing the wide spread processing capabilities of volunteer processors. In order to utilize the wide spread resources, failure options cannot be ignored. In this paper they give the detailed implementation of fault tolerance techniques, and also propose a modified forwarding mechanism which forwards the request to the next hop from which more receivers are available, the main contribution of this paper is the

implementations of fault tolerant techniques using any casting with modified forwarding mechanism and its analytical analysis for the computational grid.

Veeranjaneyulu and Srimathi [26] in their paper fault tolerance in grid computing using wade Says Many grid applications will be running in environments where interaction faults are more commonly occur between diverse grid nodes. Resources may also be used outside of organizational boundaries; it becomes iteratively difficult to guarantee that a resource being used is not malicious one. Because of the diverse faults and failure conditions developing, deploying, and executing long running applications over the grid remains a challenge. Hence fault tolerance is a primary factor for grid computing. They proposed a prototype system is designed using agents to provide service replication, reactivation and avoids the single point of failure. The agents and the workflows are provided by a common software platform called WADE.

Versweyveld [27] International science grid this week (ISGTW) in their article says today's personal computers are powerful. In fact every PC today is over 100 times more powerful than the Cray-1 machine that designed the stealth bomber in the 1970s. But, most of the time, a lot of their computational capacity goes to waste. Desktop grids can make use of this unused computational power. With more than one billion desktop computers in use, desktop grids can offer a low cost, readily available scientific resource." We are actually sitting on a huge source of computational power that is largely left unused in numbers of universities, research institutes, companies, home offices and households," said Leslie Versweyveld of the International Desktop Grid Federation (IDGF).

REFERENCES

[1] 1987. Avizienis, A., et al., (Ed.). Dependable Computing and Fault-Tolerant Systems Vol. 1: The Evolution of Fault-Tolerant Computing, Vienna: Springer-Verlag. (Though somewhat dated, the best historical reference available.)
[2] 1988. Harper, R., J. Lala, and J, Deyst, "Fault-Tolerant Parallel Processor Architectural Overview," Proc of the 18st International Symposium on Fault-Tolerant Computing FTCS-18, Tokyo, June 1988. (FTPP)
[3] 1990. Computer (Special Issue on Fault-Tolerant Computing) **23**, 7 (July). 1991. Lala, J., et. al., The Draper Approach to Ultra Reliable Real-Time Systems, Computer, May 1991.
[4] 1991. Jewett, D., A Fault-Tolerant Unix Platform, Proc of the 21st International Symposium on Fault-Tolerant Computing FTCS-21, Montreal, June 1991 (Tandem Computers)
[5] 1991. Webber, S, and J. Beirne, The Stratus Architecture, Proc of the 21st International Symposium on Fault-Tolerant Computing FTCS-21, Montreal, June 1991.
[6] 1993. Briere, D., and P. Traverse, AIRBUS A320/A330/A340 Electrical Flight Controls: A Family of Fault-Tolerant Systems, Proc. of the 23rd International Symposium on Fault- Tolerant Computing FTCS-23, Toulouse, France, IEEE Press, June 1993.
[7] 1993. Sanders, W., and W. D. Obal II, Dependability Evaluation using UltraSAN, Software Demonstration in Proc. of the 23rd International Symposium on Fault-Tolerant Computing FTCS-23, Toulouse, France, IEEE Press, June 1993.
[8] 1993. Beounes, C., et. al. SURF-2: A Program For Dependability Evaluation Of Complex Hardware And Software Systems, Proc. of the 23rd International Symposium on Fault- Tolerant Computing FTCS-23, Toulouse, France, IEEE Press, June 1993.
[9] 1994. Blum, A., et. al., Modeling and Analysis of System Dependability Using the System Availability Estimator, Proc of the 24th International Symposium on Fault-Tolerant Computing, FTCS-24, Austin TX, June 1994. (SAVE)
[10] 1994. J.H Lala, R.E Harper, Architectural Principles for Safety-Critical Real-Time Applications, Proc. IEEE, V82 n1, Jan 1994, pp25-40.
[11] 1994. Jenn, E. , J.Arlat , M.Rimen , J.Ohlsson , J.Karlsson, Fault injection into VHDL models: the MEFISTO tool, Proc. Of the 24th Annual International Symposium on Fault-Tolerant Computing (FTCS-24), Austin, Texas, June 1994.
[12] 1995. Siewiorek, D., ed., Fault-Tolerant Computing Highlights from 25 Years, Special Volume of the 25th International Symposium on Fault-Tolerant Computing FTCS-25, Pasadena, CA, June 1995. (Papers selected as especially significant in the first 25 years of Fault- Tolerant Computing.)
[13] 1995. Baker, W.E, Horst, R.W., Sonnier, D.P., and W.J. Watson, A Flexible ServerNet-Based Fault-Tolerant Architecture, Proc of the 25th International Symposium on Fault-Tolerant Computing FTCS-25, Pasadena, CA, June 1995. (Tandem)
[14] 1996. Timothy K. Tsai and Ravishankar K. Iyer, "An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems," Proc. 26th Symposium on Fault-Tolerant Computing FTCS-26, Sendai, Japan, June 1996. (FTAPE)
[15] 1998. Kropp Nathan P., Philip J. Koopman, Daniel P. Siewiorek, Automated Robustness Testing of Off-the-Shelf Software Components, Proc of the 28th International Symposium on Fault-Tolerant Computing , FTCS'28, Munich, June, 1998. (Ballista)
[16] 1998. Spainhower, l., and T.A.Gregg, G4: A Fault-Tolerant CMOS Mainframe Proc of the 28th International Symposium on Fault-Tolerant Computing FTCS-28, Munich, June 1998. (IBM).
[17] 1998. Kozyrakis, Christoforos E., and David Patterson, A New Direction for Computer Architecture Research, Computer, Vol. 31, No. 11, November 1998.