An Overview of Software Fault Tolerant Computing

K.C. Joshi
Computer Center, IASE,
M.J.P. Rohilkhand University, Bareilly
kc-joshi@indiatimes.com

ABSTRACT
Fault-tolerance is the survival attribute of the real-time software system. The software should provide correct results in the face of various failures. A major technological concern for the coming years is the ever widening gap between the demand for high quality, robust software and its supply. Software fault tolerance is basically the design faults in the computer system. Its function is to prevent system accidents, and mask out faults if possible. A characteristic of the software fault tolerance techniques is that they can, in principle, be applied at any level in a software system: procedure, process, fault application program, or the whole system including the operating system. The continuous research works are done in the field of software fault tolerant computing to make error free software. But still it is difficult to make error free software due to complexity of the system.

This paper presents a review fault tolerant computing, historic view of fault-tolerant computing, design fault tolerance, software fault tolerant techniques, software fault tolerance methods and concludes some thoughts on challenges of software fault tolerant computing.

KEYWORDS
Software Fault, Redundancy, Reliability, Design Diversity, Checkpointing.

1. FAULT-TOLERANT SOFTWARE
In real-time systems, software is the key to the performance and error-free operation of the system. Real-time software is a special class of software with certain unique characteristics - such as operation in unpredictable and asynchronous environments and response generation according to strict deadline schedules.

Reliability, safety and fault-tolerance are desirable features of real-time software. Reliability is the probability that the system will perform its intended function for a specified period of time under a set of specified environmental conditions. Safety is the probability that conditions leading to an accident do not occur whether or not the intended function is performed. In general, reliability requirements are concerned with making the system failure-free whereas safety requirements are concerned with making it accident-free. Fault-tolerance is the survival attribute of the real-time software system. The software should provide correct results in the face of various failures. A major technological concern for the coming years is the ever widening gap between the demand for high quality, robust software and its supply. This is of utmost relevance in the Indian context since there are concerted efforts underway to produce control software for life critical systems including satellite launch vehicles, high-tech combat aircraft, C-3 systems, nuclear power plants and hazardous chemical processes.

2. HISTORIC VIEW OF FAULT-TOLENT COMPUTING
Fault-tolerant computing has evolved from specialized military and communications systems to general purpose, high-availability commercial systems. The evolution of fault-tolerant computers has been well documented [4, 24]. The earliest high availability systems were developed in the 1950's by IBM, Univac, and Remington Rand for military applications. In the 1960's, NASA, IBM, SRI, the C. S. Draper Laboratory and the Jet Propulsion laboratory began to apply fault tolerance to the development of guidance computers for aerospace applications. The 1960's also saw the development of the first AT&T electronic switching systems. The first commercial fault-tolerant machines were introduced by Tandem Computers in the 1970's for use in on-line transaction processing applications [19]. Several other commercial fault-tolerant systems were introduced in the 1980’s [20]. Current commercial fault-tolerant systems include distributed memory, multi processor Tandem NonStop [21], Tolerant Eternity [22], shared-memory transaction-based systems (Sequora '23 ), “pair-and-spare” hardware fault-tolerant systems (Stratus [24], DEC VAXft 3000 [23]), and triple-modular-redundant systems (Tandem Integrity S2).

Most applications of commercial fault-tolerant computers fall into the category of on-line transaction processing. Financial institutions require high availability for electronic funds transfer, control of automatic teller machines and stock market trading systems. Manufacturers use fault tolerant machines for automated factory control, inventory management, and on-line document access systems. Other applications of fault tolerant machines include reservation systems, government databases, wagering systems, and telecommunications systems.

Vendors of fault tolerant machines attempt to achieve both increased system availability and continuous processing.
Depending on the system architecture, either processes continue to run despite failures or the processes are automatically restarted from a recent checkpoint. Some traditional systems have enough redundancy to reconfigure around failed components, but processes running in the failed modules are lost. Vendors of commercial fault-tolerant systems have extended fault tolerance beyond the processors and disks. To make large improvements in reliability, all sources of failure must be addressed, including power supplies, fans and inter-module connections.

3. SOFTWARE FAULT TOLERANCE
Software plays a crucial role in a computer system's ability to tolerate design, manufacturing, and wear-out-faults. Faults in software are typically due to problems in design or implementation, while faults in hardware can be due to design, manufacturing, wear-out, or environmental upsets. Software fault tolerance is an overview of the ways in which software design and implementation techniques can be used to detect and tolerate both software design errors and hardware faults. The development of highly reliable software necessitates more than just software fault-tolerance techniques. The development process must include rigorous application of fault avoidance approaches, which include the correct use of formal specification languages, structured programming, formal proof of correctness, and extensive testing at all levels of implementation. Design for fault avoidance is a necessary prerequisite for effective software fault tolerance [2]. Software fault tolerance addresses the issues of detecting and recovering from residual design and implementation errors in the software and detecting and recovering from wear-out and environmentally induced hardware faults.

4. DETECTION AND RECOVERY FROM SOFTWARE FAULTS
The fundamental approach to detecting software design errors is through exploiting diversity. Diversity in implementation and design can be in the form of acceptance tests, executable assertions, alternative software modules, or full diversity through designing and implementing multiple versions of the complete software by different teams of software engineers. Diversity can be captured through encoding knowledge of the expected behavior at various levels of the software and then comparing what is expected against what is observed. The encoding of knowledge can be at the level of the process outputs, intermediate results, system behavior or expected algorithm behavior. The two primary approaches to software fault tolerance that provide a complete framework for capturing diversity in both design and implementation, as well as providing formal mechanism for error detection, error containment, and recovery, are: (1) recovery blocks [1]; and (2) N-version software [2].

5. SOFTWARE FAULT TOLERANCE METHODS
5.1 RECOVERY BLOCKS
Recovery blocks, as developed by Randell [1], implement diversity in the form of acceptance tests and alternative software modules. Software is partitioned hierarchically into self-contained modules called "recovery blocks." Each recovery block validates its own operation and either returns correct results or notifies the system of an error. As illustrated as follows, each recovery block is composed of an acceptance test, the primary alternative software module, and the secondary software modules. The acceptance test is used to determine the correctness of a software module's results (error detection) and the alternative modules provide recovery from a detected error. Diversity can be captured in both the acceptance test and the secondary alternative software modules.

```
ensure <acceptance test>
by <primary module>
else by <alternate module1>
else by <alternate module2>

... ...

else by <alternate module n>
else error.
```

A recovery block is ultimately responsible for the correctness of its own output; therefore, if the acceptance test exhausts all of the alternative processing routines, a mechanism should be provided to signal that a failure condition has detected and contained.

5.2 CHECKPOINTING AND ROLLBACK
Check pointing is an important technique for recovery after error detection by means of rollback re-execution of a process. Check pointing schemes can be broadly classified as full or incremental check pointing. The former saves the entire active state space of a process while the latter saves the difference between the current and the previous state space. A check pointing scheme can be implemented at the system or application level.

Research on classic check pointing and rollback recovery has been extensive [25-28]. Graph–theoretical methods by which the programmer can decide where to insert checkpoints have been developed. The program is decomposed by the programmer into a sequence of tasks between which the checkpoints can be inserted. It is assumed that the execution time, the checkpoint time, and the recovery time of each of these tasks are known in advance. With this information, the algorithms can determine the optimal places to insert checkpoints so that the maximum checkpoint time, the expected checkpoint time, or the expected run time is minimized.

5.3. N-VERSION PROGRAMMING
The N-version programming approach to fault tolerant software has been extensively described by Avizienis [2]. N version
programming differs from recovery blocks in that it employs design diversity at the software system level through designing and implementing multiple (N) versions of the software with different teams of programmers. Instead of employing an acceptance test, N version programming utilizes voters to reach a consensus of two or more outputs among the N member versions. This approach necessarily must rely on diversity in the design to detect programming errors in the N versions of the software. If diversity is not enforced in the design and implementation, there may be an undetected or unrecoverable failure due to a single cause. Both the recovery block and N version programming approaches require a reliable execution environment for young or executing assertions and for time-efficient execution of the software modules.

6. SOFTWARE FAULT TOLERANCE TECHNIQUES:
6.1 REDUNDANCY
Redundancy consists of hardware components’ physical replication such as replicated computers. Thus, redundancy is a fundamental prerequisite for a system that either recovers from or hides failures [7]. Redundancy can be provided in two different ways called static or dynamic redundancy. In static redundancy, the idea is to use three or more modules which have the same input signals while they are all active. Their outputs are connected to a voter that compares these signals. The correct signals are then chosen by majority voting. The faulty module can be masked by 2-out-of-3 voting.

Dynamic redundancy uses less number of modules on the expense of more information processing. A minimal configuration uses 2 modules. One module is usually in operation and if it fails the standby or backup unit takes over. This requires a fault detection unit to detect the faulty situations. Simple fault detection modules use the output signal for consistency checking, comparison with redundant modules, and information redundancy in computers like parity checking or watchdog timers. The task of the reconfiguration module is to switch to the standby module from the faulty one after the fault is detected.

As in its hardware counterpart redundancy methods, software redundancy methods are used.

6.2 DIVERSITY
Design diversity in a very expensive approach, as the same software has to be developed several times, by several teams. Diversity is software redundancy where different software implementations are proposed to ensure the independence of common development errors of the redundant components.

Design diversity is a defense against “common mode” or “common cause” development errors in safety critical systems [8]. It is a system design concept that attempts to reduce the possibility of failure stemming from a common development error in one functional failure path; this can result in another functional failure path. This is accomplished by designing a functional failure path to be sufficiently different to minimize the likelihood that the error will manifest itself in another component. Faults can be generated, but successfully masked and ignored within the system.

The two major forms of software redundancy are N Self-Checking Programming and N-version programming.

N Self-Checking Programming:
A self-checking program results from adding redundancy to a program so that it can check its own dynamic behavior during execution. A self-checking software component consists of either a variant and an acceptance test or two variants and a comparison algorithm.

N-version Programming:
As we already discussed above, in an N-version software system, each module is made with up to N different implementations. Each variant accomplishes the same task, but hopefully in a different way. Each version then submits its answer to voter or decider which determines the correct answer. An important distinction in N-version software is the fact that the system could include multiple types of hardware using multiple versions of software. The goal is to increase the diversity in order to avoid common mode failures. Both redundancy and diversity increase hardware costs, weight, and power requirements for all redundant components.

7. RECOMMENDATIONS FOR SOFTWARE FAULT TOLERANCE
Some of the recommendations that can be observed during system development can be study as:

- It is expected that software fault tolerance research will benefit from this research by enabling greater predictability of the dependability of software.
- Software implemented fault tolerance is responsible for the detection, isolation and handling of faults which are visible to it and to the handling of faults detected by the operating system, the hardware and the application software.
- Some special areas relating to fault tolerant system development: i) VLSI fault physics ii) logic testing and design for test iii) system architecture iv) software correctness proofs v) robust operating system vi) reliability and performance modelling vii) fault-tolerant software through recovery blocks or design diversity.
- The attraction of fault-tolerant software is the possibility of an economic advantage over single-version software in attaining the same level of trustworthiness.
- At present, fault-tolerant software is the only alternative that can be expected to provide a higher level of trustworthiness and security for critical software units than test or proof techniques without fault tolerance.
The industry needs low cost techniques that can provide an added measure of fault tolerance and reliability to the computers used by the vast majority of users.

7. CONCLUSION
Fault tolerance always has a cost attached, in terms of development time and money and operating time and money. This cost must be weighed against the costs of system failure. N-version programming has the potential of increasing the reliability of some aspects of a program, provided that development and testing time and funding are increased to cover the extra costs. In this paper we emphasize on the technique for software fault tolerance techniques. Other areas like software fault tolerance methods, techniques are also discussed. Software fault tolerance is not a panacea for all our software problems. Since, at least for near future software fault tolerance will primarily be used in critical systems, it is even more important to emphasize that “fault tolerant” does not mean “safe”, not it cover the other attributes comprising dependability.

REFERENCES

Proceedings of the 5th National Conference; INDIACom-2011

Copy Right © INDIACom-2011 ISSN 0973-7529 ISBN 978-93-80544-00-7