

Beyond the Conventional Techniques of Software Fault Tolerance

*A low cost and unconventional technique for gaining software fault tolerance
without using N-modular redundancy in both software and hardware.*

By Goutam Kumar Saha

Computer software has rapidly become an important and indispensable element in many aspects of our daily lives. In recent years, producing reliable software for realtime control systems has become a major interest of the industrial as well as the academic world. We expect such systems to operate reliably, even under extremely severe conditions. However, no matter how thoroughly we test, debug, modularize and verify, design bugs will still plague our software.

Moreover, software often gets corrupted or experiences random errors in codes and data while running in an industrial environment. Whatever measure we may take it is very difficult to make an industrial environment free from the potential threats of various electrical transients or noises. Software failures may lead to partial or total system crashes. In some cases, for example in air traffic control, nuclear power plant control systems, or space vehicle systems, such crashes would cost money, potentially cost lives, and have other detrimental societal impacts. Our dependency on software continues to escalate. Software size and complexity continues to grow and find many new applications such as safety assurance systems for nuclear power, life support systems in health care, etc.

Owing to the critical nature of these applications, operational reliability is of paramount importance. Therefore, to achieve ultra-reliability in industrial computing, it is necessary to adopt the strategy of defensive programming based on redundancy. This is referred to as fault-tolerant software.

While industries clearly understand the need for fault tolerance, some are hesitant to put it into practice due to a) the additional cost of redundancy, b) software complexity and c) the

environment in which the system operates. However, with the current growth of software system complexity, we cannot afford to postpone the implementation of fault tolerance in critical software application areas. Redundancy is accepted as a viable approach for obtaining reliability with unreliable components. Fault-tolerant software ensures system reliability by using protective redundancy at the software level.

Fault-Tolerant Software Design Approaches

A *Recovery Block Scheme (RBS)* comprises three elements: a) one *primary module* for executing critical software functions, b) an *acceptance test* in order to test the primary module's output after each execution, and c) one set of alternate modules performing the same function of the primary module. Here, we expect that the test condition will be met by the successful execution of either the primary module or the alternate modules. When an *acceptance test* detects a primary module failure, an alternate module executes, and so on. If all the modules are exhausted or failed then the system crashes. In RBS modules are executed sequentially. This scheme is similar to the *dynamic redundancy* in hardware.

In the *N-Version Programming Scheme (NVPS)*, N-independent programs execute in parallel on identical input, and results are obtained by voting upon the outputs from individual programs. In order to ensure the development of independent program versions, we must use different algorithms, techniques, programming languages, environments and tools in each effort. In this scheme modules are executed in parallel. In critical systems with realtime deadlines, voting at program's end (as in the basic NVPS) may not be acceptable. Therefore, voting at intermediate points is a must for realtime systems.

In a *Community-Error-Recovery Scheme (CERS)*, we need to compare results at intermediate points of computation. It offers higher fault tolerance than the basic NVPS. However, this approach requires the overhead of synchronizing the various software versions at comparison points. Here, online detection is carried out by an acceptance test.

The *N-Self-Checking Version Scheme (NSCVS)* adopts intermediate voting at which each iteration is subject to an acceptance test or comparison check. Whenever a version raises an exception, the correct output can be obtained from the remaining versions and then the execution of the industrial application software continues.

In the *Enhanced-Single-Version Scheme (ESVS)*, the author suggests the use of a single version of the application software that is enriched with enhanced processing logic using various thoughtful checkpoints in order to detect errors or faults in program and data flow immediately, and, to take recovery actions. It prevents error propagation. This scheme is a low-cost and efficient solution in order to design in high fault tolerance. It does not need multiple versions of programs, rather it uses only an enhanced version [1,2,3,4] of the application program. Again, it does not need multiple computing machines. It needs only one machine in order to run the application. Therefore, it needs lower redundancy in both memory space and runtime.

Conclusion

This article has briefly described various software fault tolerance approaches. The ESVS approach demands a thorough study of the application system. System design engineers having sound knowledge of the application system will find it a very useful and economical tool while designing various industrial application systems, with built-in higher fault tolerance, dependable computing, software safety and system reliability, using minimum modular redundancy in both software and hardware. It is a very low cost and an effective solution towards designing in various software fault tolerant scientific and industrial applications.

References

- [1] Goutam Kumar Saha, "Transient Software Fault Tolerance Through Recovery", ACM Ubiquity, Vol. 4(29), Association for Computing Machines (ACM) Press, 2003, USA.
- [2] Goutam Kumar Saha, "Transient Software Fault Tolerance in Scientific Computing", Paper No-00068-2003, Journal of ACM, ACM Press, USA.

[3] Goutam Kumar Saha, " Single – Version Algorithm for Transient Fault Tolerance", paper no. TSE-0175-1003, IEEE Transactions on Software Engineering, IEEE Computer Society, 2003, USA.

[4] Goutam Kumar Saha, " Transient Software Fault Tolerance Analysis Using Algorithms", in press, International Journal on System Analysis Modelling Simulation (SAMS), Gordon and Breach, 2003, USA.

Goutam Kumar Saha (gksaha@rediffmail.com) has worked as a computer scientist for last 16 years. He worked in various research organizations including LRDE, Defence Research & Development Organisation (DRDO), Bangalore, and the Electronics Research & Development Centre of India (ER&DCI), Calcutta. His present employer is the Centre for Development of Advanced Computing (CDAC), Kolkata. He is a senior member in IEEE (USA), Computer Society of India (CSI). He is a Fellow Member in MSPI (New Delhi), IMS (Goa).