AN EFFICIENT APPROACH FOR SOFTWARE FAULT TOLERANCE IN PARALLEL COMPUTING

[1]Jashan Deep, [2]Rajiv Mahajan
[1]M.Tech Student (CSE), [2]Professor in Dept. of CSE
GIMET, Amritsar, Punjab, India

ABSTRACT
Computer software has rapidly become an important and indispensable element in many aspects of our daily lives. No one is 100% free from bugs, for producing reliable software for real-time control systems has become a major interest of the industrial as well as the academic world. It is expected such systems 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. There is two traditional methods for software fault tolerance i.e. recovery block and N- Version Programming and one proposed an efficient approach for software fault tolerance in parallel computing environment named as "Single Version Scheme". This paper shows the working of single version scheme.

KEYWORDS: *SVS, Bugs, Threads, Process, Procedures.*

1. INTRODUCTION

The only thing constant is change. This is certainly more true of software systems than almost any phenomenon,[18] not all software change in the same way so software fault tolerance methods are designed to overcome execution errors by modifying variable values to create an acceptable program state.[19] The need to control software fault is one of the most rising challenges facing software industries today. It is obvious that fault tolerance must be a key consideration in the early stage of software development.

There are different methods for software fault tolerance, among which:

- Recovery blocks
- N-version software

SVS: SVS means Single version scheme. The goal is to determine the fault occurrence in a system. The various types of Acceptance tests are used for the faults detection. These are:

- The result of a program is to be tested.
- If the result passes the test, he program continues its execution.
- A failed test indicates a fault.

WORKING OF SVS**: SVS technique relies on procedure triplication in order to bear one flawed computation for the result. The basic steps involved in this scheme are stated below[15].**

Step 1.Triplicate an application program in the form of a procedure: PI1, PI2, and PI3

Step 2.Sequentially execute: PI1, PI2 with similar inputs.

Step 3.Validate the signature-based control-flow checking and then compare the outputs say, RI1 and RI2 of PI1 and PI2 respectively.

Step 4.If both the outputs (values) are found to be same on comparison (no transient or permanent bit error has occurred or amidst *fail silent faults*), *then* application system's output is RI1 or RI2.

Step 5.If both the outputs (values) are not same, and then execute the third image PI3 with similar input.

Step 6.Validate the signature-based control-flows and compare the outputs (values) obtained from either of these application-copies that is, either RI1 and RI3, or RI2 and RI3, in order to find out equality and to output it.

Step 7.If run-time signatures of control-flows are detected as erroneous then we need to compare (or vote) all the outputs from all the three replicas of an application, and if there is an output (value) in majority, then application-system's output is the majority one only. Thus, faults in either one of the application-replicas or bit-errors in run-time signatures are tolerated by masking the erroneous output (caused by transient or permanent bit errors).

Step 8.If no output in majority, *then* application is restarted or reloaded for re-execution. In such disagreement, SVS converges to a fail-stop kind scheme.

The proposed approach is a SVS, the three copies of a procedure that is, *Proc*, *Proc_cp1* and *Proc_cp2* are sequentially executed on similar inputs, and on validating their control-flows, their outputs are compared or voted upon in order to mask an erroneous output. The working principle of the proposed approach is shown in Figure.

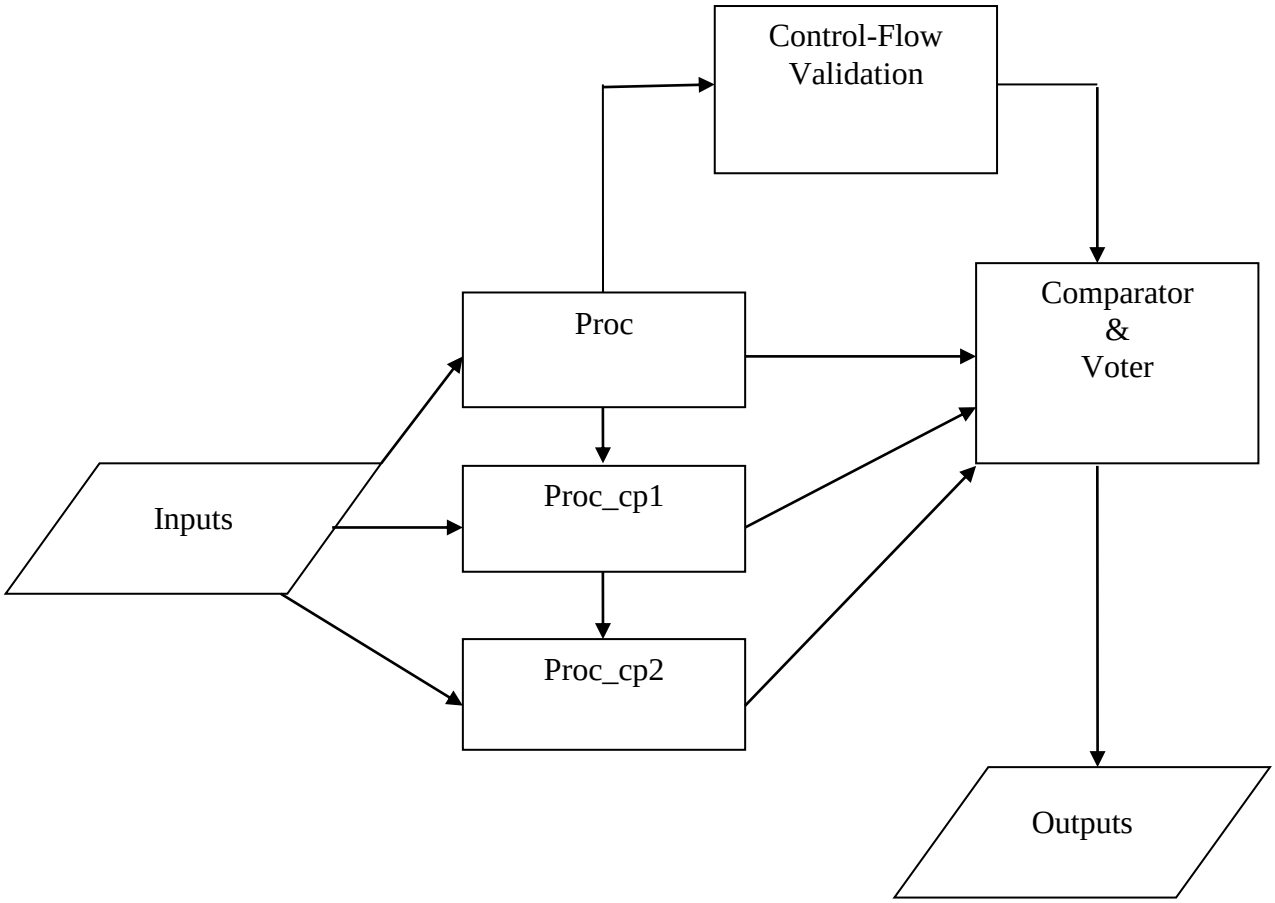

Figure 1 Schematic Diagram of a Single Version Scheme (SVS)

2. CONCLUSION

Software fault tolerance is an ability to check the system and the possibility of the system even in the presence of faults. Here SVS (Single version scheme) is discussed for the software fault tolerance. The working of a SVS uses the concept of threads which is light weighted activity, which affects the time and memory complexity of the system. The main focus is on check the working of the system for long duration in the presence of the system. The second main task is to Study and analysis the effects of complexity over fault tolerance by different sorting approaches like Bubble Sort, Insertion sort and Selection sort.

REFERENCE

[1] A. Benzekri and R. Puigjaner, "Fault Tolerance Metrics and Evaluation Techniques", 1992.
[2] Brian Randell and JieXu, "*The Evolution of the Recovery Block Concept*," University of Newcastle upon Tyne, England, 1971.
[3] Chris Inacio, "*Software Fault Tolerance*" Carnegie Mellon University18-849b, Dependable Embedded Systems, Spring 1998.
[4] D.F. McAllister and M. A. Vouk, "*Software Fault-tolerance Engineering*," Chapter 14 in Handbook of Software Reliability Engineering, Mcgraw Hill, editor M. Lyu pp. 567-614, January 1996.
[5] *K.C. Joshi* & *Durgesh Pant*, "*Software Fault Tolerant Computing: Needs and Prospects*" , ACM Ubiquity, Vol 8 issues 16 , April 2012.
[6] George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, David I. August, "*SWIFT: Software Implemented Fault Tolerance*," Departments of Electrical Engineering and Computer Science, Princeton University, NJ, June 2002.
[7] Goutam Kumar Saha, "*Software Fault Avoidance Issues*," Member ACM, Ubiquity -- Volume 7, Issue 46 (November 28, 2006 - December 4 2006).
[8] Kim, K. H., and H. O. Welch, "Distributed Execution of Recovery Blocks: An Approach for Uniform Treatment of Hardware and Software Faults in Real-Time Applications," IEEE Transactions on Computers, Vol. 38, No. 5, 1989, pp. 626-636.
[9] Laprie, J., and C., "Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures," IEEE Computer, Vol. 23, No. 7, 1990, pp. 39-51.
[10] C.V. Ramamoorthy et al., "*Software Engineering: Problems and Perspectives*," Computer, Vol. 17, No. 10, pp. 191-209, October 1984.
[11] ZaipengXie, Hongyu Sun and KewalSaluja *"A Survey of Software Fault Tolerance Techniques,"* University of Wisconsin-Madison/Department of Electrical and Computer Engineering, 2011.
[12] M. Sghairi, A. de Bonneval, Y. Crouzet, J.-J.Aubert and P. Brot *"Challenges in Building Fault-Tolerant Flight Control System for a Civil Aircraft,"* International Journal of Computer Science, Vol. 35 No.4, Nov 2008.
[13] Goutam Kumar Saha" A Single-Version Algorithmic Approach to Fault Tolerant Computing Using Static Redundancy",Clei Electronic Journal, Vol 9, No. 2, Paper 9, December 2006.
[14] Bharathi V,*" N-Version programming method of software fault tolerance: A Critical Review"*, National Conference on Nonlinear Systems & Dynamics (NCNSD)-2003.
[15] Pham Ba Quang, Nguyen Tien Dat, Huynh Quyet Thang *"Investigate the relation between the correctness and the number of versions of Fault Tolerant Software System"* World Academy of Science, Engineering and technology3 2007.
[16] Goutam Kumar Saha *"A Software Tool for Fault Tolerance"* Centre for Development of Advanced Computing, Journal of Information science and engineering 22,953-964, 2006.
[17] Goutam Kumar Saha *"A Single-Version Scheme of Fault Tolerant Computing"* Centre of Development of Advanced Computing, Kolkata, India, JCS&T, Vol. 6, No. 1, April 2006.
[18] Eckhardt, D. E., "Fundamental Differences in the Reliability of N-Modular Redundancy and N-Version Programming", The Journal of Systems and Software, 8, 1988, pp. 313–318.[19] Ray Giguette and Johnette Hassell, "Toward A Resourceful Method of Software Fault Tolerance", ACM Southeast regional conference, April, 1999.