Volume 5, Issue 9, September 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Approaches of Software Fault Tolerance Computing

Dr. K. C. Joshi
Faculty of Education and Allied Sciences
M.J.P. Rohilkhand University, Bareilly, Uttar Pradesh, India

Abstract: ***In the area of software engineering , sofware are developed by skilled programmer using a number of recently developed or recently successful innovations include libraries for fault tolearnt computing, object oriented language, remote procedure calls that are the foundation for client-server application, application programs interfaces, graphical user interface, prototyping and source level debugging.***
Fault tolerance is the set of techniques aimed at detecting, isolating and recovering from computational state that can lead to failure. Fault or bug is that produces error and error leads to failure. Error is a state of the system under investigation, a state that can bring down the whole system. Software faults are most often caused by design shortcomings that occur when a software engineer either misunderstands a specification or simply makes mistake. It is estimated that 60-90% of current computer errors are from software faults. Software faults may also be triggerd from hardwae ; these faults are usually transitory in nature, and can be masked using a combination of current software and hardware fault tolerance techniques. For industrial point of view the need for fault toleranr due to : a) the additional cost of redundancy b) software compexity c) environment in which the system operates.
Software fault tolerance is the heart of the building trustworthy sofware. Trustwothy software is stable. Trustworthy software does what it is supposed to do and can repeat that action time after time, always producing the same kind of output from the same kind of input. The study of software fault tolerance starts with the goal of making software product available to users in the face of software errors.
As software will never be free of bugs , techniques for software fault tolerance must be applied to make applications robust against such bugs. There are number of approaches for tolerating fault techniques such as recovery block sustem, N-version programming (NVP), Consensus recovery block(CRB), Distributed Recovery Block (DRB),N-self checking version progrmming(NSCP), Roll-Forward Checkpointing Scheme (RFCS), Extended Distributed Recovery Block (EDRB). In this paper we will compare the above approaches and also discuss the beneficial and shortcomings of these approaches. A continuous research is required in this area as it sophisticated and harmful in our real life.

Key words: Software Fault, Hardware Fault, Software fault tolerance,Recovery Blocks.

I. INTRODUCTION

Faults are, in turn, caused by numerous problems occurring at specification, implementation, fabrication stages of the design process. Faults due to incorrect implementation, usually referred to as design faults, occur when the system implementation does not adequately implement the specification. In hardware, these include poor component selection, logical mistakes, poor timing or synchronization. In software, examples of incorrect implementation are bugs in the program code and poor software component reuse. Software heavily relies on different assumptions about its operating environment. Faults are likely to occur if these assumptions are incorrect in the new environment. The Ariane 5 rocket accident is an example of a failure caused by a reused software component. Ariane 5 rocket exploded 37 seconds after lift-off on June 4th, 1996, because of a software fault that resulted from converting a 64-bit floating point number to a 16-bit integer. The value of the floating point number happened to be larger than the one that can be represented by a 16-bit integer. In response to the overflow, the computer cleared its memory. The memory dump was interpreted by the rocket as an instruction to its rocket nozzles, which caused an explosion.
Software becomes more complex and more significant to the overall system performance and dependability. Software costs and reliability are a major concern in the life cycle of systems. As hardware reliability has been improved and as the costs for hardware have been reduced, software faults have emerged as the most significant problem in new critical systems. Furthermore, as new applications become more complex, the reliability of the software increases in significance.

II. FAULT TOLERANCE

Fault tolerance is achieved by using some kind of redundancy. It is how to provide, by redundancy, service complying with the specification in spite of faults having occurred or occurring,The redundancy allows either to mask a fault, or to detect a fault, with the following location, containment and recovery.
Fault masking is the process of insuring that only correct values get passed to the system output in spite of the presence of a fault.
Fault detection is the process of determining that a fault has occurred within a system.
Fault location is the process of determining where a fault has occurred.

Fault containment is the process of isolating a fault and preventing the propagation of the effect of that fault throughout the system.

Fault prevention
It is how to prevent, by construction, fault occurrence. Fault prevention is achieved by quality control techniques during specification, implementation and fabrication stages of the design process.

Fault removal
It is how to minimize, by verification, the presence of latent errors. Fault removal is performed during the development phase as well as during the operational life of a system. During the development phase, fault removal consists of three steps: verification, diagnosis and correction.

Fault forecasting
It is how to estimate, by evaluation, the presence, the creation and the consequences of errors.Fault forecasting is done by performing an evaluation of the system behavior with respect to fault occurrences or activation. The evaluation can be *qualitative*, that aims to rank the failure modes or event combinations that lead to system failure, or quantitative, that aims to evaluate in terms of probabilities the extent to which some attributes of dependability are satisfied, or coverage.

Fault Tolerance in distributed system
Distributed system is a collection of Independent computer that appear to its user as a single coherent system. Like online railway reservation system, air traffic central , Internet Banking are distributed system where human-to -human communication is easier. It allows many user access to a common computing resource i.e. Resource sharing.
In distributed system a single fault may lead to huge loss of money and even human livens.In such situation ,inclusion of fault tolerance technique is essential. Fault tolerance techniques enable system to perform tasks in the presence of faults. There are high chance that more than one fault may occur in distributed system.

Hardware and Software Fault:
Hardware component faults may be permanent ,transient or intermittent but design fault will always be permanent. However, it should be remembered that faults be of any of these classes may result in errors that persist within the system. According to new reports coming out from Asia, the newly unveiled Samsung Galaxy S6 are SGS6 edge hard sets both suffer from pretty grievous fault in the form of a display error that stops them property ready touch inputs around the bezel .The fault is allegedly a hardware , rather that software. Software defect can comes corruption of data or control flow information. Since such corruption is due to Software defects, one would expect the frequency at which it happens depends heavily on the quality of the software.

Fault Tolerant computing approaches:
A number of fault tolerant computing approaches was studying which are based on the passed research work and some experimental work done by the researchers . Some of them are discussed here and their comparison is also discussed.

III. RECOVERY BLOCKS

This scheme for software fault-tolerance can be regarded as analogous to hardware fault-tolerant "stand-by sparing." The basic recovery blocks scheme is one of the two original designs diverse software fault tolerance techniques. As from figure-1, recovery blocks uses an Acceptance Test (AT) to accomplish fault tolerance.As the system operates, checks are made on the acceptability of the results generated by each software component. Should one of these checks fail, a spare component is switched on to take the place of the faulty component. The spare component is, of course, not merely a copy of the main component. Rather it is of independent design, so that there can be the possibility that it can cope with the circumstances that caused the main component to fail. Of course, recovery blocks satisfy this general description. The deadline mechanism [CAMP79] and dissimilar backup software are also included here because of their similarity with recovery blocks.

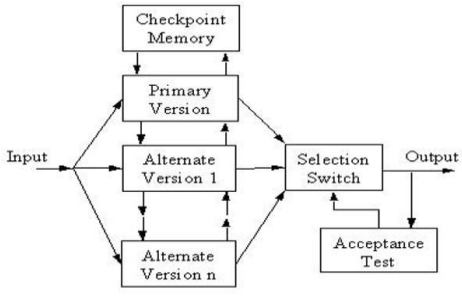

Figure 1. Recovery Block model

ensure <acceptance test>
by <primary module>

else by <alternate module1>
else by <alternate module2>
.
.
.
else by <alternate module n>
else error.

To perform a recovery block operation, the primary alternate (which corresponds to the block of the equivalent conventional program) is performed. An acceptance test is run to determine whether the alternate has performed acceptably. If the primary version fails to *complete* or fails the acceptance test, the system state is restored to that *current* just before entry into the primary version and an alternate version is performed. If the primary version passes the acceptance test, all alternates are ignored and the statement following not pass the acceptance test, the entire recovery block is considered to have failed, so the block in which it is embedded fails to complete. Recovery is then attempted at that level.

N-Version Programming

N-versions serially in a single processor (N-Version serial), or execute each version in parallel on N loosely-coupled processors (N-Version parallel) . N-Version programming [A. Avizienis and L. Chen] is a multi-version technique in which all the versions are designed to satisfy the same specification and the decision mechanism (DM) examines the results and selects the "bets" result, if one exists. There are many alternative DM available for use with NVP. N-version programming is analogous to static redundancy for hardware fault-tolerance. N>2 functionally equivalent, independently generated versions are provided input from a system supervisory program called a driver. The driver executes a comparison algorithm on the versions' results.

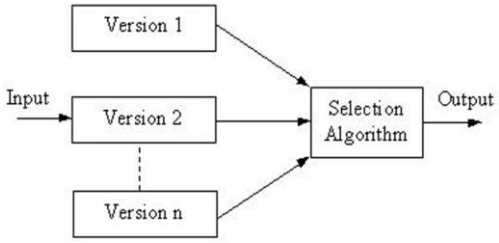

Figure 2. N-Version programming model

As from the figure-2 The basic elements of the N-version programming approach are:

* the *initial specification* - this is the specification of the functionality which is desired by the software;
* *N software versions* - software modules which all are independently generated from the initial specification; and
* a *decision mechanism* - a mechanism which decides what the final result of the computations will be using the results from the N versions as input.
* a *supervisory program* - this is a software structure used to drive the N versions and the decision mechanism.

IV. CONSENSUS RECOVERY BLOCK

The consensus recovery block method requires the development of n independent versions of a program, an acceptance test and a voting procedure.

The consensus recovery block (CRB) technique is a combination of NVP and Recovery Block, at first NVP runs and if it fails to produce the correct result, recovery Block runs and produces the correct result[4]. Initially, all versions execute and submit their outputs to a voting procedure. Since it is assumed that there are no common faults, if to more versions agree on one output, that output is designated as correct. If there is no agreement, that is, the versions supply incorrect outputs or multiple correct outputs, then a modified recovery block is entered. Each version is examined sequentially in a predetermined order. The output of the "best S' version is subjected to the acceptance test. If that output is judged acceptable, it is treated as a correct output, and system execution continues.

If, on the other hand, the output is not accepted, the next best version's output is subjected to the acceptance test. This process continues until an acceptable output is found, or the N outputs are exhausted. Notice that there is no requirement for input state recovery since all versions execute in a parallel fashion.as in an N-Version programming system. This technique might be more aptly termed consensus multi-version software.

Self-Configuring Optimal Programming:

SCOP (Self-configuring optimal programming) [Bondavalli et al. 1993; Xu et al. 1993] is another attempt to combine some techniques used in RB and NVP in order to enhance efficiency of software fault tolerance and to eliminate some inflexibilities and rigidities. This scheme organizes the execution of software variants in phases, dynamically configuring a currently active variant set, so as to produce acceptable results with the relatively small effort and to make the efficient use of available resources. The control can be parameterized with respect to the level of fault tolerance, the amount of available resources and the desired response time.

Since highly dynamic behaviour can cause complexity of control and monitoring, a methodology for devising various instances of SCOP is developed by simplifying the on-line process at the price of the complex off-line design.

Distributed Recovery Block (DRB):

Distributed recovery block (DRB) technique, is the distributed version of Recovery Block technique in which several recovery blocks are implemented in several systems. the only difference between these blocks is the priority of modules [K. H. Kim].

The distributed recovery block (DRB) was introduced in 1984 by Kim in [Kim84]. The aim was to integrate software and hardware fault tolerance into one single structure since it may be very difficult to distinguish between software faults and hardware faults. The DRB is as the name suggests based on the recovery block scheme.

In the DRB system, one processor executes the primary alternate while the other executes the secondary. If an error is detected in the primary results, the results from the secondary are immediately available. The

dependability analysis of both systems is identical. The main difference between the both schemes is that both the primary and the alternate modules are replicated and are resident on two or more separate nodes interconnected by a network. Software faults are hopefully coped with in the traditional recovery block fashion, and hardware faults are handled by the distribution of computation among several nodes.

N Self-Checking Programming (NSCP)

After a study of fault tolerant systems and how fault tolerance was implemented in these systems, Laprie et al. presented the findings as the N Self-Checking Programming (NSCP) approach in [Lap87] and [Lap90]. In this approach, the system is divided into several *self checking components* comprised of different variants (equivalent to alternates in RB and versions in NVP) of the software. These components execute in parallel. The hardware fault tolerance architecture related to NSCP is active dynamic redundancy. A self-checking program uses program redundancy to check its own behavior during execution. It results from either the application of an AT to a variant's results or from the application of a comparator to the results of two variants.

Roll-Forward Check-pointing Scheme (RFCS)

A scheme for multiprocessor environments was introduced by D.K. Pradhan and N.H. Vaidya in [Pra94a][Pra94b]. RFCS scheme provides a mechanism for identifying the faulty module and recovering , in most likely cases, without the overhead of rollback. The objective was to achieve performance of a Triple Modular Redundancy (TMR) system using duplex redundant systems. An RFCS system uses a multiprocessor organisation consisting of a pool of active processing modules (PM) and either a small number of spare modules or active modules that have some spare processing capacity. Each PM consists of a processor and volatile storage (VS) and has access to a stable storage (SS). All processing modules are assumed identical and also that they have access to the stable storage of other modules. There is also a reliable checkpoint processor (CP) available. This processor detects module failures by comparing the state of each pair of processing modules that perform the same task (i.e. a duplex system).

Extended Distributed Recovery Block (EDRB):

The extended distributed recovery block (EDRB) was introduced by Hech et al. in 1991 [Hec91a] [Hec91b]. EDRB is a fault tolerant real time distributed system for critical process control applications. The underlying fault tolerant mechanisms are based on extensions to the DRB which is in turn based on classical recovery blocks and real time extensions. There are several kinds of nodes in an EDRB structure. Nodes responsible for control of the process and related systems are called *operational nodes*. These nodes are often critical since they provide some kind of real-time service and store non-recoverable state information. Therefore they are redundant. The operational nodes are clustered in sets of two, so called *node pairs*. The members of a node pair exchange periodic status messages called *heartbeats*. A node in a node pair can recover from failure in its companion node if the malfunction is declared as a part of the heartbeat message. Should one of the nodes in a pair stop sending its heartbeats, the other node will request a confirmation of the first nodes failure from a second kind of node called the *supervisor*. The supervisor is a single node that confirms the absence of a heartbeat, arbitrates inconsistent states in operational node pairs, and logs all status changes and failures. The supervisor node may be important to the EDRB but it is not critical, since its failure only impacts on the ability of the system to recover from failures requiring confirmation or arbitration. If no other failures occur, the EDRB will continue to function without the supervisor.

Comparison:

On the basis of earlier study and research papers the following comparison is short listed and given below some of them :

1. In Recovery Block and NVP-M techniques, when the reliability of AT Module or Voter equals to 1, we can have a system with the arbitrary ability to return right results without considering the reliability of versions, all we need is the large enough number of versions.
2. NVP with consensus voting may be marginally more reliable than CRB with consensus voting when the AT reliability is low, or when AT and program failures produce identical and wrong (IAW) results.
3. Most of the time, CRB with consensus voting is more reliable than NVP with consensus voting.
4. The CRB technique has been found to be surprisingly robust in the presence of high inter version correlation.

5. In general, when the AT is not of very high quality, CRB tends to outperform NVP and to perform competitively with the Recovery Block technique.
6. When there is failure independence between variants and a zero probability of identical and wrong (IAW) answers, CRB is always superior to NVP and to Recovery Block.
7. When there is a very high failure correlation between variants, CRB is expected to outperform NVP (given the same voting strategy). only when the variants that do fail coincidentally return different results. (The CRB-AT would then be invoked.)
8. When the probability of IAW results is very high, CRB is not superior to NVP. The NVP does not perform well either in this situation.
9. For n = 3, CRB with majority voting has reliability equal to or better than the reliability of NVP with majority voting.
10. For n = 5, with a lower n-tuple reliability, NVP with consensus voting performs almost as well as CRB.
11. In comparing reliability analysis of the three different architectures, DRB performed better than NVP which in turn was better than NSCP. DRB also enjoyed the feature of relative insensitivity to time in its reliability function. This comparison, however, had to be conditioned by the probability of decider failures. In the safety analysis NSCP became the best in the by-case parameters, followed by NVP and then DRB. In the by-frame data the order was reversed again.
12. As from Zaipeng, Hongyu Sun and Kewal Saluja the comparison of the above techniques given below .

Software Fault tolerance Techniques		Abbreviation	Error Processing Techniques
Design Diversity	Recovery Blocks	RcB	Error detection by acceptance test (AT) and backward recovery
	N-Version Programming	NVP	Vote
	Distributed Recovery Blocks	DRB	Error detection and result switching, can be detected by AT, or by comparison
	N Self-Checking Programming	NSCP	Error detection by AT and forward recovery
	Consensus recovery Block	CRB	Vote then AT
	Acceptance Voting	AV	AT first then Vote
Data Diversity	Retry Blocks	RtB	Acceptance test and backward recovery
	N-Copy Programming	NCP	Run the same process concurrently or sequentially
	Two-Pass Adjusdicators	TPA	Use both data and design diversity to handle multiple correct results (MCR)

V. FUTURE WORK

The N-version programming approaches employ design diversity in an effort to increase the reliability of software. Correctness of system will increase when the number of versions increases. But in fact when increasing the number of versions, the system will be more complex and the reliability and the availability of system may decrease. So with a specific system, we need a proper number of versions that best satisfies the requirements of the reliability, the availability and Correctness. The EDRB is a practical application of distributed fault tolerance in actual process control settings. As such, it provides a vehicle for additional knowledge and experience needed to transform the potential into technically viable system. The RFCS scheme requires that at most five check pointing states be stored in the reliable storages when a failure occurs while FCS (forward recovery check-pointing) scheme requires that at most three check pointing states be stored for each case.

The new software fault tolerance techniques could be either improvement versions of some traditional techniques. The new research is done in area like adaptive n-version systems, fuzzy voting, abstraction, parallel graph reduction, rejuvenation.

VI. CONCLUSION

The main objective of this paper has been to introduce some of the fundamental concepts in fault tolerance. In fulfilling this objective, we have introduced the main issues related to N-version programming, recovery block, CRB,DRB, SCOP,NSCP, RFCS,EDRB of fault-tolerant systems.

From our research papers and theoretic calculation we have given following some conclusions:

- Correctness of fault tolerant software is covariant with the number of versions, but not linear.
- In Recovery Block technique, when the reliabilities of all versions equal to 1, we also can have a system with the arbitrary ability to return right results without considering the reliability of the AT Module, all we need is the large enough number of versions.
- In NVP techniques with Weighted Average Voter, we can't make any relation between Correctness and the number of versions. This ability depends on the way to determine the weights of versions.
- CRB with majority voting is more stable and is at least as reliable as NVP with majority voting.

- The advantage of using CRB may be marginal in high failure correlation situations or where the AT is of poor quality.
- CRB performs poorly in all situations where the voter is likely to select a set of identical and wrong (IAW) responses as the correct answer.
- It is noted (by Vouk and McAllister) that, given a sufficiently reliable AT or binary output space or very high inter version failure correlation, all schemes that vote may have difficulty competing with Recovery Block technique.

The application of all of these techniques is relatively new to the area of fault tolerance.Each technique will need to be tailored to particular applications. This should also be based on the cost of the fault tolerance effort required by the customer. For betterment of fault tolerant computing the new research is continued.

REFERENCES

[1] Martin Hiller "Software Fault-Tolerance Techniques from a Real-Time Systems Point of View - an overview", Technical Report No. 98-16**,** November 1998.

[2] Pradhan D.K., Vaidya N.H., "Roll-Forward Check-pointing Scheme: A Novel Fault-Tolerant Architecture", IEEE Transactions on Computers, Vol. C-43, No. 10, pp. 1163-1174, 1994.

[3] Pradhan D.K., Vaidya N.H., "Roll-Forward and Rollback Recovery: Performance-Reliability Trade-Of"f, Proceedings of the 24th International Symposium on Fault-Tolerant Computing, pp. 186-195, 1994.

[4] Scott R.K., Gault J.W., McAllister D.F., "The Consensus Recovery Block", Proceedings of the Total System Reliability Symposium, pp. 74-85, 1983.

[5] Hecht H., "Fault-Tolerant Software for Real-Time Applications", ACM Computing Surveys, Vol.8, No. 4, pp. 391-407, December 1976.

[6] A. Avizienis and L. Chen, 'On the Implementation of N-Version Programming for Software Fault Tolerance During Execution', Proceedings of the IEEE COMPSAC'77, November 1977, pp. 149 – 155.

[7] Algirdas Avizienis, "The Methodology of N-Version Programming", in R. Lieu, editor, Software Fault Tolerance, John Wiley & Sons, 1995.

[8] L. L. Pullum, "Software fault tolerance techniques and implementation": Artech House Publishers, 2001.

[9] K. H. Kim, "The Distributed Recovery Block Scheme," M. R. Lyu(ed.), Software Fault Tolerance, pp. 192-198, 1995.

[10] Sima Emadi, 'Software Fault Tolerance, Chapter 4: Recovery Block- RcB, Computer Engineering Department, Islamic Azad University, Maybod Branch,Yazd, Iran .

[11] Kim K.H., "Distributed Execution of Recovery Blocks: An Approach to Uniform Treatment of Hardware and Software Faults", Proceedings of the 4th International Conference on Distributed Computing Systems, pp. 526-532.

[12] Hecht M., Agron J., Hecht H., "A New Low Cost Distributed Fault Tolerance Architecture for Nuclear Reactor and Other Critical Process Control Applications", Proceedings of the 21st International Symposium on Fault-Tolerant Computing, pp. 462-469, 1991.

[13] Hecht M., Agron J., Hecht H., "A New Low Cost Distributed Fault Tolerance Architecture for Process Control Applications", Proceedings of the Southeastcon, Vol. 1, pp. 253, 1991.

[14] Laprie J.C., et al., "Hardware- and Software-Fault-Tolerance: Definition and Analysis of Architectural Solutions", Proceedings of the 17th International Symposium on Fault-Tolerant Computing, pp. 116-121, 1987.

[15] Laprie J.C., et al., "Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures", IEEE Computer, Vol. C-7, No. 7, pp. 39-51, 1990.

[16] Zaipeng Xie, Hongyu Sun and Kewal Saluja, "A Survey of Software Fault Tolerance Techniques", University of Wisconsin-Madison/Department of Electrical and Computer Engineering 1415 Engineering Drive, Madison WI 53706 USA.

[17] J. Xu, A. Bondavalli and F. DiGiandomenico, "Software fault tolerance: dynamic combination of dependability and efficiency", Tech. Report, no. 442, Computer Sci., Univ. of Newcastle upon Tyne, 1993.

[18] [Bondavalli et al. 1993] A. Bondavalli, F. DiGiandomenico and J. Xu, "Cost-effective and flexible scheme for software fault tolerance," Computer Syst. Sci. & Eng., no. 4, pp.234-244, 1993.

[19] D.F. McAllister and R. Scott, "Cost Modeling of Fault Tolerant Software", Information and Software Technology, Vol 33 (8), pp. 594-603, October 1991.

[20] M.A., Vouk, Caglayan, A., Eckhardt D.E., Kelly, J., Knight, J., McAllister, D., Walker, L., "Analysis of faults detected in a large-scale multi-version software development experiment," Proc. DASC '90, pp. 378-385, 1990.

[21] P.A. Lee and T. Anderson. "Fault Tolerance: Principles and Practice", Second Edition, Springer-Verlag, 1990.

[22] B. Randell, "System structure for software fault tolerance," IEEE Trans. Soft. Eng., vol. SE-1, no. 2, pp.220-232, 1975.

[23] R.K. Scott, J.W. Gault and D.F. Mcallister, "The consensus recovery block," in Total Sys. Reli. Symp., pp. 74-85, 1985.

[24] R.K. Scott, J.W. Gault and D.F. Mcallister, "Fault tolerant software reliability modeling," IEEE Trans. Soft. Eng., vol. SE-13, no. 5, pp.582-592, 1987.

[25] 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",International Journal of Computer Science and Engineering 1;1 © www.waset.org Winter 2007 .

[26] G. Latif-Shabgahi, A. J. Hirst, and S. Bennett "A novel family of weighted average voters for fault-tolerant computer control systems" , 2004.

[27] Rinsaka, K. and Dohi, T. (2005): "Behavioral Analysis of a Fault-Tolerant Software System with Rejuvenation", Autonomous Decentralized Systems, 2005. ISADS 2005. Proceedings, 4-8 April 2005, pp. 159 – 166.

[28] K.C. Joshi & Durgesh Pant, 'Software Fault Tolerant Computing: Needs and Prospects' , ACM Ubiquity, Vol 8 issues 16 , April 24-April 30 , 2007 .

[29] Jorg Kienzle, 'Software Fault Tolerance Implementing N-Version Programming', McGill, COMP-667 - Implementing N-Version Programming, © 2009.

[30] Joshi, K.C. , "Techniques of Software Fault Tolerance, Proceedings of the 4th National Conference; INDIACom-2010 Computing For Nation Development, February 25 – 26, 2010 .

[31] [Camp79] R.H. Campbell, K.H. Horton and G.G. Belford, "Simulations of a fault-tolerant deadline mechanism," in *9th Int. Symp. Fault-Tolerant Comput.*, pp. 95-101, Madison, 1979.

[32] Kuochen Wang & Chien-Chun Wang, "A Cost-Effective Forward Recovery check-pointing Scheme in Multiprocessor Systems", Journal of Information Science and Engineering 16, 65-80 (2000).

[33] Avizienis A.,"Fault tolerant systems", IEEE Transactions on Computers, Vol.C-25, No.12, pp. 1304-1312, 1976.

[34] Avizienis A., Laprie J.C., "Dependable Computing: From Concepts to DesignDiversity", Proceedings of the IEEE, Vol. 74, No. 5, pp. 629-638, 1986.

[35] Laprie J.C. (ed.), "Dependability: Basic Concepts and Terminology",Dependable Computing and Fault-Tolerant Systems Series, Vol. 5, Springer Verlag, 1992

