International Journal of Electrical and Computer Engineering (IJECE)
Vol. 6, No. 3, June 2016, pp. 1031 ~ 1037
ISSN: 2088-8708, DOI: 10.11591/ijece.v6i3.9041

Review of Software Fault-Tolerance Methods for Reliability Enhancement of Real-Time Software Systems

Anjushi Verma, Ankur Ghaartan, Tirthankar Gayen
Computer and Systems Sciences, Jawaharlal Nehru University, India

Article Info

Article history:

Received Sep 16, 2015
Revised Mar 7, 2016
Accepted Mar 20, 2016

Keyword:

Fault tolerance
Real time systems
Reliability
Software

ABSTRACT

Real time systems are those systems which must guarantee to response correctly within strict time constraint or within deadline. Failures can arise from both functional errors as well as timing bugs. Hence, it is necessary to provide temporal correctness of programs used in real time applications in addition to providing functional correctness. Although, there are several researches concerned with achieving fault tolerance in the presence of various functional and operational errors but many of them did not address the problem concerned with the timing bugs which is an important issue in real time systems. As for real time systems, many times it becomes a necessity for a given service to be delivered within the specified time deadline. Therefore, this paper reviews the existing approaches from the perspective of real time systems to analyse the shortcomings of these approaches to present a versatile and cost effective approach in the presence of timing bugs for providing fault tolerance to enhance the reliability of the real time software applications.

Corresponding Author:

Anjushi Verma,
Computer and Systems Sciences,
Jawaharlal Nehru University, India.

1. INTRODUCTION

A real-time system is one that must produce a correct result within a specified time deadline. According to Hermann Kopetz, "A real time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced" [1]. Some examples of real time systems are aircraft control, oven temperature controller and over-temperature monitor in nuclear power station, etc.Failures in real time systems can arise from both functional errors as well as timing bugs. Hence, it is necessary to provide temporal correctness of programs used in real time applications in addition to providing functional correctness. Although, there are several researches concerned with achieving fault tolerance in the presence of various functional and operational errors but many of them did not address the problem concerned with the timing bugs which is an important issue in real time systems. As for real time systems, many times it becomes a necessity for a given service to be delivered within the specified time deadline. Therefore, this work analyses the shortcoming of the existing approaches with respect to real time systems and presents a versatile, cost effective approach in the presence of timing bugs for providing fault tolerance to real time software applications. "Software fault tolerance is the ability of computer software to continue its normal operation despite the presence of system or hardware faults." Avizienis [2] in 1977 defined Software Fault Tolerance as "*the management of faults originating from defects in design of the software*."

Reliability has always been an important quality attribute of software systems especially for mission and safety critical software systems because in such systems severity of consequences resulting from failures is very high. According to ANSI, "*Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment.*" Software is becoming more and more complex

with the passage of time to cope with the emerging application requirements. More complex a software is, more difficult it is to assure its reliability. A complex software has a large number of states (unlike the hardware), so it is not practically possible to completely test a software. Irrespective of the amount of testing one does; it is usually difficult to assure that the final software product is fault free. In order to achieve failure free operation of software, one develops some mechanism to handle those faults which remain in the system even after its development. Software fault tolerance is such a mechanism which can be used to deal with the remaining faults after development of system. Since real-time systems need to be highly reliable and as it is not always possible to ensure the development of highly reliable system therefore they are prime candidates for the inclusion of fault tolerance techniques.

2. RELATED BACKGROUND

Several techniques are currently in use to achieve fault tolerance in software like N-Version Programming, Recovery Blocks (RcB), N-Self Checking Programming, N-Copy Programming, Retry Blocks (RtB), Adaptive N-Version Systems, Rejuvenation, Fuzzy Voting, etc, yet these techniques may not be very suitable for real time systems. Software fault tolerance techniques are mostly based on traditional hardware fault tolerance. The goal of software fault tolerance techniques is to allow the system to function properly in the presence of software faults remaining in the system after completing the development of software. Unlike hardware just redundancy is not enough to deal with software faults; some form of diversity is also required along with redundancy [3].

N-Version Programming (NVP) is one of the software fault tolerance techniques based on design diversity. Elmendorf in 1972 suggested the concept of NVP and later in 1977–1978 Avizienis and Chen developed it [4]. To achieve fault tolerance, in NVP technique a decision mechanism (DM) and a forward recovery mechanism is used. At least two variants of a program which are independently designed and functionally equivalent are developed from the common specifications. All these variants called versions are run in parallel and results produced by each version is passed to the DM. Decision Mechanism selects the best result after examining all the results. Nowadays various alternative decision mechanisms are available for use with NVP. In 1974 Horning, et al. introduced Recovery Blocks (RcB) [5]. The basic RcB scheme comprises of an executive, an acceptance test, primary and alternate try blocks (variants). Based on the acceptance tests (AT) result RcB selects a variant result and pass it to the output during program execution. By using the primary alternate (or try block) the RcB technique will initially try to ensure the AT (e.g., a test is passed based on the acceptability of a result of an alternate). If the primary algorithm's result did not pass the AT, then n-1 alternatives will be tried (attempted) until an alternate's results pass the AT. An error occurs when any of the alternates can not pass the AT [6]. Laprie, et al developed N-Self-Checking Programming (NSCP) which is a design diverse technique. A self-checking program uses program redundancy to check its own behavior during execution. The NSCP hardware architecture comprises of four components grouped in two pairs in hot standby redundancy, where one software variant is supported by each hardware component. NSCP software contains a comparison algorithm and two variants or an AT and one variant on each hardware pair [7].

Data diversity as a complementary software fault tolerance strategy to design diversity was proposed by Ammann and Knight, 1987 [8]. Data diversity based techniques implements diversity at the input data. To achieve data diversity every data diversity based technique uses a data re-expression algorithm (DRA). DRA produces diverse input data sets that are logically equivalent. The performance of data diversity is highly dependent on the data re-expression algorithm. The problem associated with data diverse approaches is that all applications can not employ data diversity because it is not possible to find an effective Data re-expression algorithm for every application. Ammann and Knight developed Retry Blocks (RtB) which is one of the data diverse software fault tolerance techniques. AT and backward recovery to achieve fault tolerance is used by RtB technique. Later Ammann and Knight also developed N-Copy Programming (NCP) as a data diversity technique. The NCP technique uses a forward recovery and a decision mechanism to accomplish fault tolerance. At least two copies of a program and at least one data re-expression algorithm is used by NCP. System inputs are passed to a DRA in order to generate logically equivalent input data sets. Using input the re-expressed data the copies execute in parallel. A decision mechanism works to examine the results of the copy executions and if there exists a best result, it selects it. NCP is considered as a data diverse complement of NVP.

The new Software Fault Tolerance techniques are Fuzzy Voting, Byzantine Fault Tolerance, Adaptive N-Version Systems and Graph Reduction. Kanoun, K., et al. [9] considered modified classical N-Version systems by incorporating in each version an individual weight factor. It considers an adaptive approach to model and manage different quality levels of the versions. This weight factor is then incorporated in the voting procedure, i.e. for the deviation behavior of the individual version the voting is

based on a weighted counting of the number of monitored events. In order to deal with transient software failures caused by software aging software rejuvenation is a novel approach, which can be considered as a proactive and preventive solution to counteract the aging phenomenon. Rejuvenation Involves occasional stoppage of the running software, restarting it and cleaning of the internal state. Cleaning of the system's internal state may involve flushing kernel tables of operating system, garbage collection and to reinitialize the internal data structures, etc. Hardware reboot is well known example of rejuvenation. Because of the difficulty in justification of the extra cost of up-front development the traditional fault tolerance techniques are usually avoided in the development of mission critical systems. In the year 1996 Robert J Kreutzfeld, et. al. [10] presented a methodology called Data Fusion Integrity Process (DFIP) as an alternative for the traditional software fault tolerance techniques in order to deal with high sunk cost of development. This is a simple and effective technique for the development of fault tolerant mission critical systems. A DFIP implementation includes one recover, one detection, and the report method. The recover method's implementation realizes biggest cost saving in DFIP. The modular approach of DFIP makes it easier to implement at any time. Many software fault tolerance techniques are available but the relative effectiveness of different techniques remains unclear. S. Garnaik, et al., [11] in 2013 presented an approach for reliability enhancement of software programs by minimizing the overflow errors. They handled the problem of overflow because of large size integer input. The presented approach says that instead of storing the large size integer input in primary int data type; store it in the linked list data structure. Every node of this linked list contains two digits and pointer to next node. Authors present technique for addition and multiplication operation of large integer input using linked list. This approach provides tolerance to the faults occurring due to the use of large size data to the extent that as long as the free memory is available for allocation there won't be any failure due to use of large size data.

Hence, it was broadly observed that N-Version Programming both in the presence and absence of data diversity is not applicable for situations in which distinct multiple solutions exist. Majority voting for N-Version Programming may result in incorrect decision when majority of outputs are incorrect. Design diversity may not be very effective if similar kind of faults get detected in various versions. Implementation of effective design diversity may incur high cost due to the development of many versions. Hence it is not a cost effective approach for real time systems. Though implementation of data diversity based N-Copy Programming using data re-expression algorithm is a cost effective approach, yet it is not very useful as it is difficult to obtain an effective data re-expression algorithm for various real time system. The effectiveness of various fault tolerance techniques varies with variations in systems. Therefore there is no universal fault tolerance technique which is effective for all real time systems. There is no methodology to assess the relative effectiveness of different fault tolerance techniques. Recovery Block Technique is not suitable for real time systems due to it's serial nature of execution. From the survey, it has been found that there are several works concerned with achieving fault tolerance in the presence of various functional and operational errors but many of them did not address the problem concerned with timing bugs which is an important issue in real time systems. As for real time systems, many times it becomes a necessity for a given service to be delivered within the specified time deadline. Therefore the main objective is to develop a versatile cost effective approach in the presence of timing bugs for providing fault tolerance to real time software applications. Hence, a cost effective approach in the presence of timing bugs has been developed for providing fault tolerance to real time software applications.

3. THE PROPOSED APPROACH

In accordance with the proposed approach [12], the entire software system is divided into subsystems. Risk assessment is done on various tasks for each subsystem to identify those tasks which are more critical. From the task dependency analysis deadline is obtained for each subsystem. When a subsystem is executed, both the original program and trained neural network (TNN) module is executed simultaneously for the critical tasks of that subsystem. If there is any incomplete/missing data then the prediction module is used to provide the missing data. There is a constant polling to check whether the output is available from the original program. If the output is available from the original block before the time (***deadline – k)*** (where ***k*** is the reasonable amount of time required for completing the operations concerned with delivering the appropriate output for subsequent processing and ***deadline*** corresponds to the deadline time for the task completion) then it is forwarded further for subsequent processing else if the output is available from the TNN block then it is forwarded for subsequent processing. Otherwise, the subsystem moves to the safe mode and sends appropriate signal for subsequent processing. The delay in producing the output from the original program may depend on lot of factors like task dependency, resource sharing, unavailability of the required data etc. The detailed steps concerned with this approach is specified in algorithm ***fault_tol (system)***

The proposed algorithm is specified in the following steps:-

Algorithm **fault_tol** (system)
{
Step 1: Divide the real time system into subsystems based on tasks.
Step 2: Filter the task based on risk assessment.
Step 3: Obtain deadline for each sub system based on the task dependency analysis.
Step 4: For each sub system repeat step 5 to step 10.
Step 5: Start the timer.
Step 6: If the input data set is complete then pass the input to Trained Neural Network (TNN) block and original program.

Else pass incomplete input to prediction module and then pass predicted complete input data to TNN block and original program.
Step 7: Start execution of TNN block and original program simultaneously.
Step 8: If timer < (Deadline – k)
Step 8.1: If the output of the original program is available then forward it for subsequent processing

Else go to step 8.1.
Step 9: If timer > = (Deadline – k)
Step 9.1: If the output from the Trained Neural Network block is available then forward it for subsequent processing

Else go to step 10.
Step 10: Move the subsystem to the safe mode and send appropriate signal for subsequent processing.
}

Where, **k** is the reasonable amount of time required for completing the operations concerned with delivering the appropriate output for subsequent processing deadline corresponds to the deadline time for the task completion.

4. IMPLEMENTATION DETAILS

Presently, work is in progress for the implementation of the Trained Neural Network block for various functions. As already stated that for achieving the fault tolerance in real time systems, it is necessary to provide temporal correctness along with providing functional correctness. This work has focused on the aspect of providing the temporal correctness to the real time software systems. Some basic operations like addition and subtraction using the neural network had already been implemented for 32 bit integers. For training, a full adder logic has been used. The input data used for training the neural network is shown in Table 1.

Table 1. Binary input combination for full adder

Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

For training the neural network for this basic addition operation back propagation was used. At the input layer there are 2 nodes, at hidden layer there are 4 nodes and at the output layer finally there are 2 nodes.

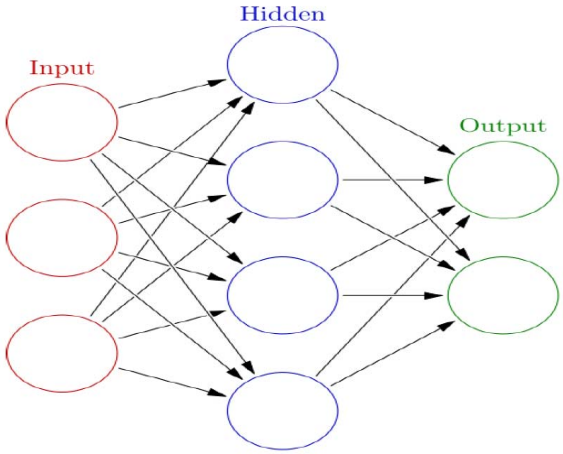

Figure 1. Neural network used for the implementation

The final weight matrix of the edges between various layers is as follows:

Table 2. Final learned weight matrix of input to hidden layer edges

	1	2	3	4
1	-2.25661097601	5.53403644389	-4.31645140220	6.03423933659
2	-2.58656210241	6.71371721611	4.59349480724	2.15647765277
3	-3.42512067886	5.82928004774	5.98654538189	-2.02944103020

Table 3. Final learned weight matrix of hidden to output layer edges

	1	2
1	-1.15939605063	6.32874007295
2	10.48179616772	6.33829272745
3	9.45494337461	1.15671512472
4	9.49956424497	2.74877040583

After successful training of this neural network block it has been tested for various integer inputs within the range. Some of the results are as follows:

demo(150,-600,F_W_IH,F_W_HO,F_B_H,F_B_O)
ans = -450
demo(200,400,F_W_IH,F_W_HO,F_B_H,F_B_O)
ans = 600

It is found to perform the operation correctly. The prediction block is being implemented using Markov model and various other steps of this approach are being applied to various real-time applications to obtain the results.

5. DISCUSSIONS

Failures due to timing bugs may depend on several factors based on task dependencies, resource sharing, operational profile, etc. If it is considered that the failures caused by timing bugs occur randomly, then the failure rate due to timing bugs is constant. Let the constant failure rate due to timing bugs be **k**. If the failure rate of the actual software (in the presence of timing bugs) is **z(t)**. Then the failure rate of fault-tolerant programs (avoiding timing bugs based on the proposed approach) is $z(t) - k$. If **z(t)** is an exponentially decreasing function [11] (during the testing/debugging phase) then the plots in Figure 2 for actual and fault-tolerant programs are obtained. If $z(t)$ is a linearly decreasing function (during the testing/debugging phase) then the plots in Figure 3 for actual and fault-tolerant programs are obtained.

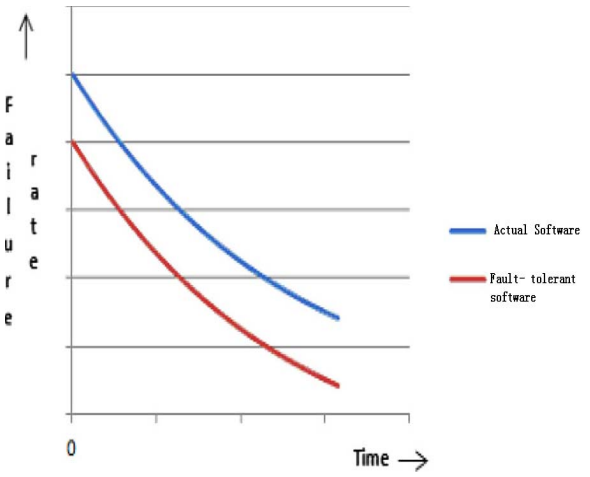

Figure 2. Failure rates for exponentiallydecreasing z(t)

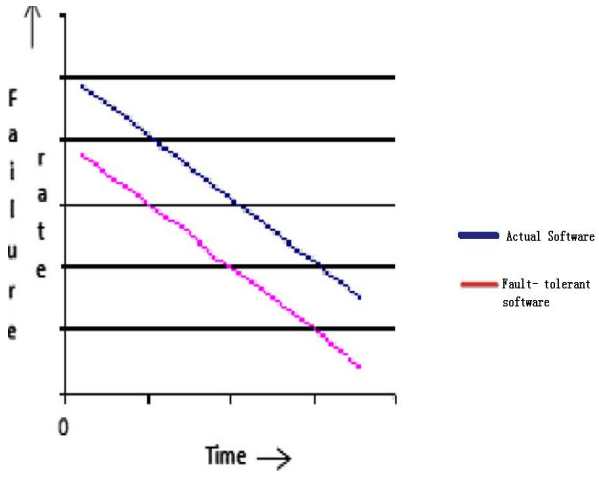

Figure 3. Failure rates for linearly decreasing z(t)

Hence, it is evident that the failure rate of the fault-tolerant programs (obtained from the proposed approach) are always less than that of the original programs when timing bugs are present. The failure rates are equal when there are no timing bugs. Since, it is observed that the failure rate is less for fault-tolerant programs (obtained from the proposed approach) as compared to the original program in the presence of timing bugs. Hence, it can be inferred that the reliability value of the given original program is enhanced in the presence of timing bugs by using the proposed approach to obtain the fault-tolerant program. Table 4, shown the comparison of the proposed approach with other approaches for their suitability with respect to real-time systems.

Table 4. Comparison of the suitability of various approaches with respect to real-time systems

Techniques	Suitability for real time systems	Reasons
N-Version Programming	Poor	o Not suitable when distinct multiple solutions exist. Majority voting for N-Version Programming may result in incorrect decision when majority of outputs are incorrect. o Design diversity may not be very effective if similar kind of faults get detected in various versions o The decision mechanism may not be adequate enough to handle timing bugs for providing temporal correctness.
Recovery Blocks[9]	Poor	o It may not be suitable for real time systems due to it's serial nature of execution. o No mechanisms to handle timing bugs for providing temporal correctness.
N-Self Checking Programming	Fair	o Design diversity may not be very effective if similar kind of faults get detected in various versions o No mechanisms to handle timing bugs for providing temporal correctness.
N-Copy Programming	Fair	o Since, it is difficult to obtain an effective data re-expression algorithm for various real time systems. o No mechanisms to handle timing bugs for providing temporal correctness.
Retry Blocks	Poor	o It may not be suitable for real time systems due to it's serial nature of execution o No mechanisms to handle timing bugs for providing temporal correctness.
Proposed approach[14]	Good	o Provides mechanism to handle timing bugs and unavailability of data, thereby providing temporal correctness, in addition to providing functional correctness for real time software systems.

From, Table 4, it is found that the proposed approach is found to be more suitable as compared to other approaches for real time software systems.

6. CONCLUSION

A real-time system should process information and produce a response within a specified time. Real time systems are time critical and their correctness depends on both the correctness of output and their timeliness. In real time systems failure may cause very severe consequences. Since it is not always possible to ensure the development of highly reliable real time systems hence they are prime candidates for the inclusion of fault tolerance techniques. In this paper, reviews of the existing approaches from the perspective of real time systems to analyze the shortcomings of these approaches and finally proceeds to present a versatile and cost effective approach in the presence of timing bugs for providing fault tolerance, thereby enhancing the reliability of the real time software applications. The proposed approach needs to be implemented for various real time software applications. This approach is helpful in providing the temporal correctness for real time systems in the presence of timing bugs thereby providing fault tolerance and making the application more reliable.

ACKNOWLEDGEMENTS

The authors would like to thank all the staffs of SC & SS, JNU for their support behind this work.

REFERENCES

[1] H. Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications," *Willey*, 2nd edition, 2006.
[2] Anderson T. and Knight J. C., "A Framework for Software Fault Tolerance in Real-Time Systems," *IEEE Transactions on Software Engineering*, vol/issue: 9(3), pp. 355-364, 1983.
[3] X. Zaipeng, *et al*., "A Study Of Software Fault Tolerance Techniques," *1415 Engineering Drive, Madison WI 53706 USA*, 1998s.
[4] L. Chen and A. Avizienis, "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," *Proceedings of FTCS- 8, Toulouse, France*, 1978, pp. 3–9.
[5] J. J. Horning, *et al*., "A program structure for error detection and recovery," in *Oper. Syst., Proc. Int. Symp (Lecture Notes in Computer Science), vol. 16, E. Gelenbe and C. Kaiser, Eds. Berlin: Springer-Verlag*, 1974, pp. 171-187.
[6] J. Gray, "Why Do Computers Stop and What Can Be Done About It?" *Proc. Fifth Symp. Reliability in Distributed Software and Database Systems, Jan*, 1986, pp. 3-12.
[7] A. Avizienis and J. P. J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," *IEEE Computer*, vol/issue: 17(8), pp. 67–80, 1984.
[8] P. E. Ammann and J. C. Knight, "Data Diversity: An Approach to Software Fault Tolerance," *IEEE Transactions on Computers*, vol/issue: 37(4), pp. 418–416, 1988.
[9] K. Kanoun, *et al*., "Reliability growth of fault tolerant software," *IEEE Transactions on Reliability*, vol/issue: 42(2), 1993.
[10] J. K. Robert and R. E. Neese, "A Methodology For Cost Effective Software Fault tolerance For Mission-Critical Systems," *15th AIAA/IEEE Digital Avionics Systems conference, Atlanta, GA*, 1996, pp. 19-24.
[11] S. Garnaik, *et al*., "Reliability Enhancement of Software by Minimizing the Overflow Errors," *International Journal of Systems Assurance Engineering and Management, Springer*, vol/issue: 5(4), pp. 724-730, 2014.
[12] Ghartaan A. and Gayen T., "Analysis and proposition of fault-tolerance model for real time software systems," in *the proceedings of International Conference on Computing, Communication, Electrical, Electronics, Devices & Signal Processing, India, Lankapalli, India*, pp. 353-357, *Discovery*, vol/issue: 44(202), pp 62-67, 2015.