International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 4 (2015) pp. 11521-11533

http://www.ripublication.com

An Neuro-Fuzzy Approach Based Voting Scheme For Fault Tolerant Systems

[1]D. Uma Devi, [2]P. Seetha Ramaiah
[1]Department of CS and SE, Andhra University, Vishakapatnam, Andhra Pradesh 530003, India, umadevi.odm@gmail.com
[2]Department of CS and SE, Andhra University, Vishakapatnam, Andhra Pradesh 530003, India

Abstract

A neuro-fuzzy voting scheme which is an enhancement of the inexact majority voter for fault tolerant systems is introduced by this paper. Fuzzy sets provide this information in a more comprehensible/natural form, and handle uncertainties at varied levels. A fuzzy voting scheme where the problem of choosing a 9x value or soft range for voter threshold is smoothed is proposed, considering the impact of uncertainties. A neuro-fuzzy approach, combining connectionist and fuzzy approaches merits are soft computing's key component. It aims to bring neuro-fuzzy models under a unified soft computing framework and rule extraction and rule refinement are included in a rule generation perspective. Rules learned and generated for fuzzy reasoning/control are considered.

Keywords: Fault Tolerant Systems, Voting Algorithms, Fuzzy Logic

Introduction

Fault-tolerant computing is building computing systems that operate satisfactorily when faults are present. A fault-tolerant system can tolerate one/more fault-types including operator errors, transient, intermittent or permanent hardware faults, software and hardware design errors or externally induced upsets/physical damage [1]. Most hardware fault-tolerant designs were directed to building computers that automatically recovered from random faults in hardware components. Techniques for this involved partitioning a computing system into modules which are fault-containment regions. Each module is backed by protective redundancy so that others assume its function if the module fails. Special mechanisms detect errors and implement recovery. Two general hardware fault recovery approaches were used: fault masking and dynamic recovery [2].

Fault masking is a structural redundancy technique that masks faults completely in redundant modules [3]. Many identical modules execute same functions and their outputs remove errors due to a faulty module. Dynamic recovery is necessary when one copy of a computation is running at a time (or two unchecked copies, in some cases) involving automated self-repair. Similar to fault masking, a computing system is partitioned into modules backed up by spares as a protective redundancy. In dynamic recovery, special mechanisms detect faults in modules, switch out a faulty module, switch in a spare, and instigate software actions like rollback, initialization, retry and restart to restore and continue computation. Special hardware with software is required to do this in single computers, while in multi-computers this is managed by other processors.

Efforts to attain software that tolerates software design faults (programming errors) used static/dynamic redundancy approaches similar to those for hardware faults [4]. One approach, N-version programming, uses static redundancy as independently written programs (versions) that perform similar functions, and their outputs are voted at special checkpoints [5]. Here, data being voted may not be exactly similar, and a criterion is used to identify and reject faulty versions and determine a consistent value (through inexact voting) that all good versions can use. An alternative dynamic approach is based on the recovery blocks concept. Programs are partitioned into blocks and acceptance tests executed after each block. When an acceptance test fails, a redundant code block is executed.

Fault tolerance is based on redundancy which is accomplished through software, hardware, information and time. This paper is about software fault tolerance. There were many examples of software fault tolerance in literature with fault tolerance being achieved through varied software versions that are independently developed, compiled through independent compilers, or run on different platforms. These redundant software versions were intended to be functionally equal but independent in all other aspects, to achieve greater system reliability and hence fault independence. In the presence of different outputs from varied redundant software versions, a methodology to select correct output is required. Conventional methodologies are output classification based. Such methodologies choose a correct answer based on different conditions. This traditional, numerical values based concept, considers all outputs classification into pair-wise disjoint subsets known as outputs partitions. Elements in such subsets are, “equal” if not identical within certain tolerance. Methods not based on output classification use varied Schneider’s convergence functions which are fault tolerant midpoint, median, average (egocentric or not) and weighted average [6].

Fault masking is a primary approach to improve/maintain safety-critical systems normal behaviour [7]. Voting algorithms arbitrate between results of redundant modules in fault-tolerant systems. Voting algorithms are implemented in hardware/software depending on application characteristics and type of voter selected.

Voting algorithms identify variant results in agreement selecting one as voter output. Many voting algorithms were used in commercial applications. The n-input majority voter produces a correct result with (n+1)/2 voter inputs matching each other. In no majority cases, voter generates an exception flag, detected by the system

supervisor to move system to a safe state. The formalised plurality voter implements m-out-of-n voting, where m is less than a strict majority. Majority and weighted average voters were used in such applications to provide error/fault-masking capability. The former ensures a good safety level and the latter an availability level.

Two traditionally used voting algorithms are the majority and the weighted average voters., An inexact majority voter [8] In its general form produces a correct output if most of its inputs match each other; if, they are within an application-specific interval of each other. In no majority cases, the voter generates an exception flag detected by system supervisor to drive system to a safe state.

Weighted average voter calculates its redundant input values weighted mean [9]. It is useful in applications like clock synchronisation in distributed computer systems, pattern recognition and sensor planes where a result should be generated in every voting cycle. Weights are predetermined or adjusted dynamically. Calculated weights, wi, compute voter output, $z = \Sigma wi \, .xi \, / \, \Sigma wi$ where xi values are voter inputs and z the voter output. Standard majority and weighted average voters are examples of 2 distinct voting algorithms groups. One has a high safety level but with a low availability level and the other with a low safety level and a high availability level. In exact majority, voters need a particular application-specific 'voter threshold' value, whereas weighted average voters cannot produce a benign output when there is no agreement between voter inputs. Neither voter type can cope with voter inputs associated uncertainties [9].

Triple modular redundancy (TMR) is a common fault masking where circuitry is triplicated and voted [10]. Voting circuitry can be triplicated so that individual voter failures can be corrected by voting process. A TMR system fails when 2 modules in a redundant triplet create errors leading to an invalid vote. Hybrid redundancy is a TMR extension where triplicated modules are backed up with additional spares, to replace faulty modules - allowing more fault tolerance. Voted systems need more than three times the hardware as non-redundant systems, but they their computations continue without interruption when a fault occurs, allowing usage of existing operating systems.

N-version programming tolerates residual software design faults by independently developing multiple program versions from a common specification. Though it was adopted in some mission-critical applications, this approach's effectiveness is an open question. The issue is how to predict final reliability and estimate fault correlation between multiple versions [11].

Originating from social sciences, voting is a popular system combination technique in various engineering disciplines, specially in highly dependable (safety-critical, highly reliable, and high available systems), distributed systems and pattern recognition. Voting can be applied at different planes in highly dependable systems like [12]:

- At sensor level to fuse data from replicated sensors;
- At actuator level, as used in x-by-wire systems and space shuttle;
- at control level, where 3 hardware modules perform same control function to produce a single output as used in Tandem Integrity S2 Computing System and safety-critical PLC;

- At software level, where 3 software programs perform same control procedures to produce one output as used in SIFT;
- At memory and bus levels of control systems, as used in Tandem Integrity S2 Computing System; and
- At individual critical components level as in military and aerospace FPGAs.

Associated with each choice are varied trade-offs like reliability improvement, cost, safety improvement and execution time. Voters were used in the realisation of specially designed highly reliable nodes in distributed computing systems. Voters maintain consistency among replicated data to ensure database reliability and to manage replicated data in distributed (database) systems. Other voting algorithms application areas include shape and pattern recognition, object classification and heuristic knowledge handling [12].

The problem with conventional equivalence relation based on fixed tolerance is that it fails to enable adaptable comparisons. So, there is no graduate comparison for different version outputs "closer" than fixed threshold. This leads to incorrect decisions in certain cases. Crisp equivalence relation's fuzzy extension proposed provides specific numerical data relaxation. This paper introduces fuzzy voting scheme which is an enhancement of inexact majority voter where the basic problem of choosing a 9x value or a soft range for voter threshold is largely smoothed, and uncertainties impact is considered. Also, governing fuzzy rules assign zero weight value for voter inputs in complete disagreement voting cycles resulting in a benign output (a safe output) for the voter, increasing its safety level. This article presents a neuro-fuzzy rule generation algorithm. Rule generation from artificial Neural Networks (NN) is becoming popular due to its ability to provide insight to the user about symbolic knowledge in the network.

Related Works

Use of fuzzy set theory for computing the final voter output, among agreed voter inputs of an inexact voter was proposed by Kim et al., [13].where a transitive fuzzy equivalence relation was defined and incorporated in conventional consensus and maximum likelihood voting systems with telling effect. But, use of fuzzy set theory to arbitrate between redundant values is new.

Use of parallel algorithms on EREW shared-memory systems by Karimi et al., [14] to present a new voter generation – known as Parallel Plurality Voter (PPV) – ensuring plurality voter extension without enlarging calculations suiting large-scale systems with optimal processing time. As parallel algorithms have high processing speed and suit large scale systems, they achieve a new parallel plurality voting algorithm using (n/log n) processors on EREW shared-memory PRAM. The new proposed algorithm's asymptotic analysis demonstrated its time complexity of O(log n) less than time complexity of sequential plurality algorithm, i.e. Ω (n log n).

surveyed Different existing weighted average voting algorithms and their merits and demerits or limitations were surveyed by Singamsetty and Panchumarthy [15] based on which a new History based weighted Voting algorithm with Soft Dynamic Threshold was proposed. Results of the novel voting algorithm for TMR system are

compared with existing voting algorithms with the new one ensuring almost 100% Safety when 2 of 3 modules are error free. It also gives better results for one error free module. Novel voter also gives better results for multiple error conditions with all modules having errors.

A new weighted average voting algorithm based on NN capable of improving system reliability rate was introduced by Zarafshan et al., [16]. Results proved that the neural weighted average voter increased reliability 116.63% in general and 309.82%, 130.27% and 9.37% for large, medium and small errors respectively in comparison to weighted average, and 73.87% in general and 160.44%, 83.59% and 7.52% for large, medium and small errors respectively when compared to median voter.

A random troika i.e., a subset of 3 replicas as an effective combination of fault-tolerant and secure computing was presented by Kwiat et al., [17]. Majority rule had high fault tolerance but low security while random dictator was the opposite. The random troika strikes a medium combining fault tolerance attributes of majority rule with a degree of random dictators security. Due to its versatility, random troika should be the top choice of voting algorithms for high-assurance computing applications. In the voting systems context, joining reliability and security can be reliably and securely done but only with understanding when and how different voting systems should be applied.

A new voting scheme based on fuzzy set theory introduced by Latif-Shabgahi and Hirst [12] softened harsh behaviour of the inexact majority voter in the neighbourhood of 'voter threshold' and handled uncertainty and multiple error cases in the region defined by fuzzy input variables. The voter assigns a fuzzy difference value to every voter inputs pair based on numerical distance. A set of fuzzy rules determines one fuzzy agreeability value for every individual input describing how well it matches other inputs. Each voter's agreeability is then defuzzified to give a weighting value for that input which determines its contribution to voter output. Weight values are used in weighted average algorithm to calculate voter output. The voter is experimentally evaluated from the safety point of view and availability and compared to inexact majority voter in a TMR structured framework. Fuzzy voter ensures correct outputs (higher availability) than inexact majority voter with small/large errors, less incorrect outputs (higher safety) than inexact majority voter in the presence of small errors, and less benign outputs compared to inexact majority voter.

An extension to the fuzzy voting scheme via incorporating Interval Type-2 (IT2) fuzzy logic proposed by Linda and Manic [18] allows for an improved handling of uncertain assumptions about distributions of noisy/erroneous inputs essential for correct design of a fuzzy voting scheme. The new voter design features robust performance when uncertainty assumptions change over time. The IT2 fuzzy voter architecture was compared to average voter, inexact majority voter, and T1 fuzzy voter by using a refined experimental harness. Results proved improved availability, safety and reliability of IT2 fuzzy voting scheme.

Materials and Methods

A voting algorithm specifies how a voting result is obtained from input data and is the basis for a voting network (hardware) or voting routine (software). Fuzzy voter uses fuzzy logic to generate weights to calculate a weighted average voter output, Figure 1 shows basic structure of a 3-input fuzzy voter.

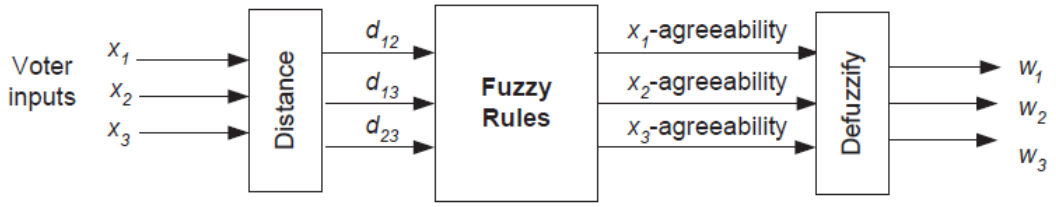

Figure 1: Three-input fuzzy voting unit

The approach's first step requires definition of a fuzzy difference variable to describe every inputs pair f to the voter, For every pair xi and xjwith numerical distance dij, based on triangular membership functions seen in Figure 2, we define a fuzzy difference variable represented by a set of membership grades μA (dij) where A: {small,medium, large}. Symmetrical sets are used, requiring 2 parameter values to be specified.Based on numerical difference between two inputs, a non-zero membership grade is assigned to one or two fuzzy sets defined for corresponding fuzzy difference variable. Triangular fuzzy membership functions are used for convenience, This definition was a natural progression from the simpler soft voter described in [27] where a ramp function is found in place of traditional hard (discontinuous) threshold seen conventional inexact majority voters.

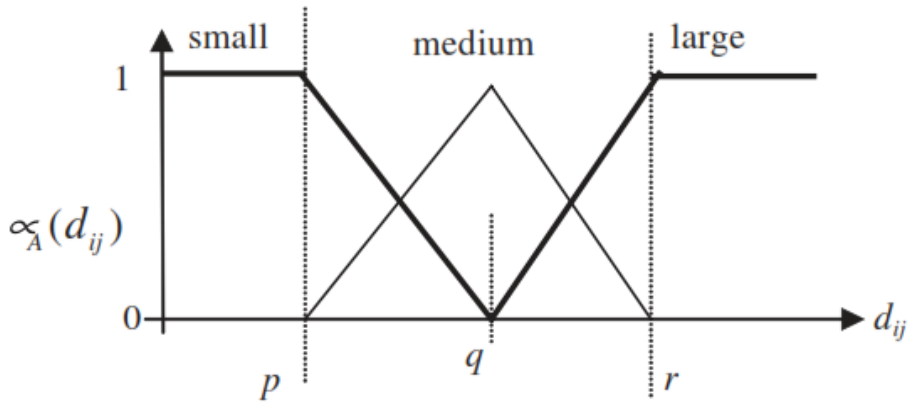

Figure 2: Variable membership functions, μ_A (dij) where A: {small,medium, large}.

The variables and fuzzy membership functions are defined as follows:

Difference between two voter-inputs: $d_{ij} = |x_i - x_j|, i \neq j.$ (1)

Symmetry: r − q = q − p where p, q, and r are real numbers, and p □ q □ r (2)

$$\mu_{\text{small}}\ d_{ij} = \begin{cases} 1 : d_{ij} \le p, \\ \dfrac{q - d_{ij}}{q - p} : p < d_{ij} \le q, \\ 0 : q < d_{ij}, \end{cases} \tag{3}$$

$$\mu_{\text{medium}}\ d_{ij} = \begin{cases} 0 : d_{ij} \le p, \\ \dfrac{d_{ij} - p}{q - p} : p < d_{ij} \le q, \\ \dfrac{r - d_{ij}}{r - q} : q < d_{ij} < \text{r}, \\ 0 : \text{r} \le \text{d}_{ij}, \end{cases} \tag{4}$$

$$\mu_{\text{large}}\ d_{ij} = \begin{cases} 0 : d_{ij} \le q, \\ \dfrac{d_{ij} - p}{r - q} : \text{q} < d_{ij} \le r, \\ 1 : r < d_{ij}. \end{cases} \tag{5}$$

In an m-way voter, there are m(m − 1)/2fuzzydifference variables. Each difference calculation results in a non-zero membership value being assigned to one or two fuzzy sets defined for that variable.

The present study proposed neuro-fuzzy rule generation algorithms. The term rule generation includes rule extraction and rule refinement.Rule extraction is extracting knowledge from ANN, using network parameters. Rule refinement, pertains to extracting refined knowledge from ANN initialized using crude domain knowledge. Rules learned and interpolated for fuzzy reasoning and fuzzy control can be also considered under rule generation.

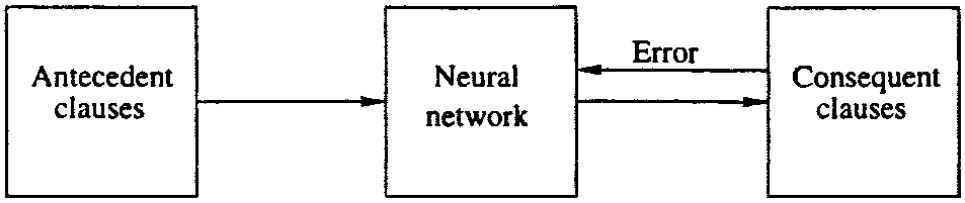

Figure 3: Neural network implementing fuzzy logic

A block diagram illustrating the process is seen in Figure 3. The input layer includes fuzzy and crisp cell groups while output is modeled by fuzzy cell groups alone. Crisp cell groups are represented by m cells taking on two values in {(+1, +1… +1), (-1,-1… -1)}.Fuzzy cell groups use binary m-dimensional vectors, each taking on values in {+1, -1}. Linguistic relative importance terms like veryimportantandmoderately important are allowed in a proposition; domain experts also assign linguistic truth values like completely true, true, possibly true, unknown, possibly false, false, and completely false depending on output values. Provision is kept, for modeling the belonging of a pattern to more than one class using different linguistic truth values. Extraction of fuzzy IF–THEN production rules is ensured using a top–down traversal involving analysis of node activation, bias and the associated link weights.

Input vector components include membership values to overlapping partitions of linguistic properties low, medium, and high corresponding to every input feature providing scope to incorporate linguistic information in training and testing phases of the models, increasing their robustness in tackling imprecise/uncertain input specifications. An n-dimensional feature space is decomposed into overlapping sub-regions corresponding to 3 primary properties low, medium, and high. Though there is a corresponding dimension and cost increase, this is offset by specific gains achieved. The scheme enables models use more feature space local information and is suitable in handling overlapping regions and nonlinear decision boundaries. Output decision is provided regarding class membership values. Ambiguous or uncertain vectors contribution to weight correction is automatically reduced. The trained network's connection weights constitute the knowledge base for the problem being considered. When a test pattern's partial information is presented at input, the model infers its category or queries the user for relevant information in relative importance order (decided from learned connection weights). The network can justify its decision in rule form with antecedent/consequent parts produced in linguistic and natural terms. Antecedent clauses are from a trained network by backtracking along maximum-weighted paths, while consequent part is generated through a certainty measure.

The hidden layer generates an internal representation of the relationship, producing the consequent's possibility distribution at the output. The model functions as an inference engine with every small sub network learning a rule's functional input–output relationship.

Results and Discussion

Details of software voters experimental test harness used in this work are explained. The experimental test harness, seen in Figure 4, simulates a TMR system [19] comprising an input data generator, replicator, 3 saboteurs (to inject errors to replicated input data), voter, and a comparator. In each test cycle, the input generator produces one notional correct result. This sequence simulates redundant modules generated identical correct results. Notional correct result copies are presented to every saboteur, in each cycle. Saboteurs are individually programmed to introduce

module errors, based on chosen random distributions. In a set of tests one, two or three saboteurs are activated to simulate module result errors on voter inputs. All saboteurs' outputs are subjected to examined voter, and voter output is compared by cycle notional correct value by the comparator.

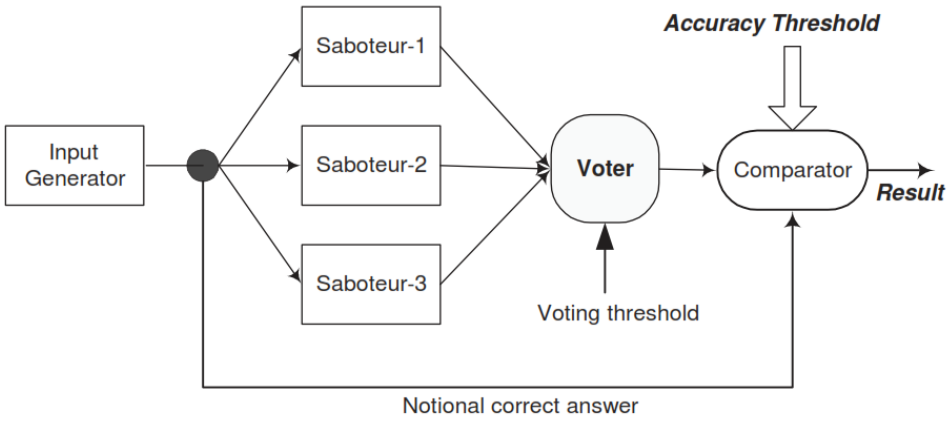

Figure 4: Experimental harness It is assumed

1. That voting is inexact;
2. That all voters perform correctly. This assumption is due to the fact that voting algorithm is usually a simpler program than modules it monitors;
3. That all voters are used in a cyclic system where there is some relationship between correct results from one cycle to next (controlling e fuel supply to an engine);
4. That faults cause errors whose symptoms appear to voter as numerical input values perturbed by varying amounts;
5. "Notional correct result" is known at any voting cycle.
6. A comparator checks for agreement between notional correct result and voter output under test at any voting cycle.
7. An output is correct if voter commits to an output value less than accuracy threshold away from notional correct result; an output value being greater than or equal to same threshold is an incorrect result; a benign answer finally, is a distinguished output reporting that no value was reliably generated. Benign outputs occur when voter cannot arbitrate between inputs to give an output, as in a case where there is disagreement between input pairs.
8. An accuracy threshold, s, is used, in comparator, to determine if distance between notional correct result and voter output is within acceptable limits. A voter result with a distance from notional correct answer less than accuracy threshold is a correct output, otherwise it is considered an incorrect output.
9. Issues associated with ensuring inputs synchronisation to voter are ignored.

Input data with sinusoidal trajectory, and fixed sample rate (0.1 second) feeds saboteurs and the comparator. Both consensus threshold and accuracy threshold parameters are set to a fixed value 0.5. Random errors having a uniform distribution

from interval [−δ +δ] are injected into saboteurs to simulate permanent (multiple) module errors which are chosen because there is relatively little published work on voters behaviour in such circumstances. But, error cases produce catastrophic consequences and must be considered. The saboteurs output is presented to voter under test. In every voting cycle voter output, z is compared to an input data copy, xo. Based on numerical distance between z and xo, voter output is interpreted as correct, incorrect or benign. For each voter, results of n system run (n is selected 10^4) are classified. Thus, n_c correct results, n_{ic} incorrect outputs, and nbbenign results are collected. The data is then used to evaluate and compare selected voters. Two performance measures are defined as:

1. *Safety (S)*: Safety measure is defined as: S = (1- n_{ic} / n). Thus, S ϵ [0 1] and ideally S =1.
2. *Availability (A)*: Availability of a voter is defined as A=n_c/n. Thus, Aε[0, 1] and ideally A=1. Each performance criteria is measured versus changes of *error-to-signal ratio.*

A voter's performance depends on many voter-specific and environmental parameters. The proposed neuro-fuzzy voter scheme's performance is compared to fuzzy voter and majority voter. Figure 5 and 6 show the different voter methods Safety and availability.

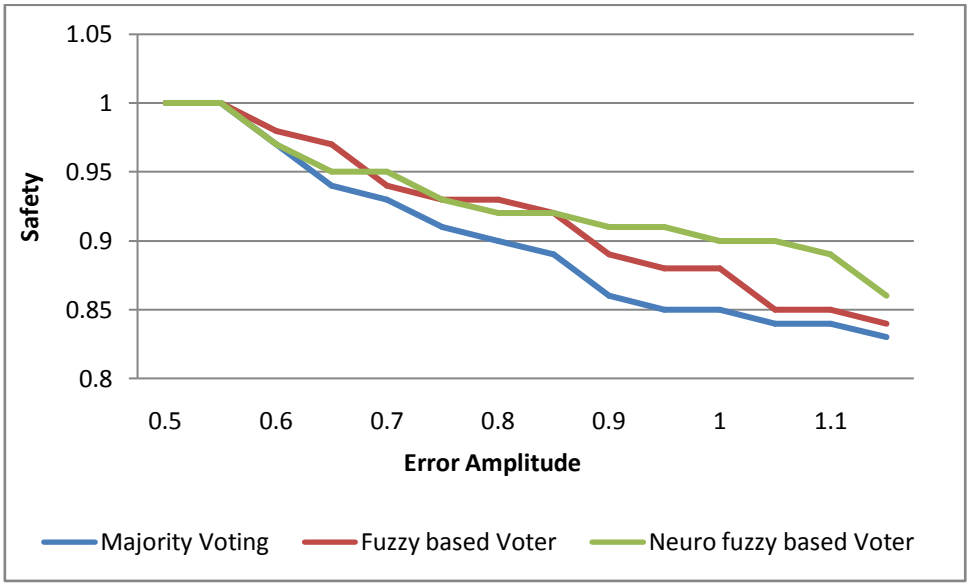

Figure 5: Safety of Voters

The proposed neuro-fuzzy voter shows better safety behaviour compared to the standard majority and fuzzy voters. When compared to majority voter, the proposed neuro-fuzzy method improves safety by 1.06% to 6.9% when the error amplitude is more than 0.65. The proposed neuro-fuzzy method improves safety by 1.03% to 5.71% when the error amplitude is more than 0.7 when compared to fuzzy voter.

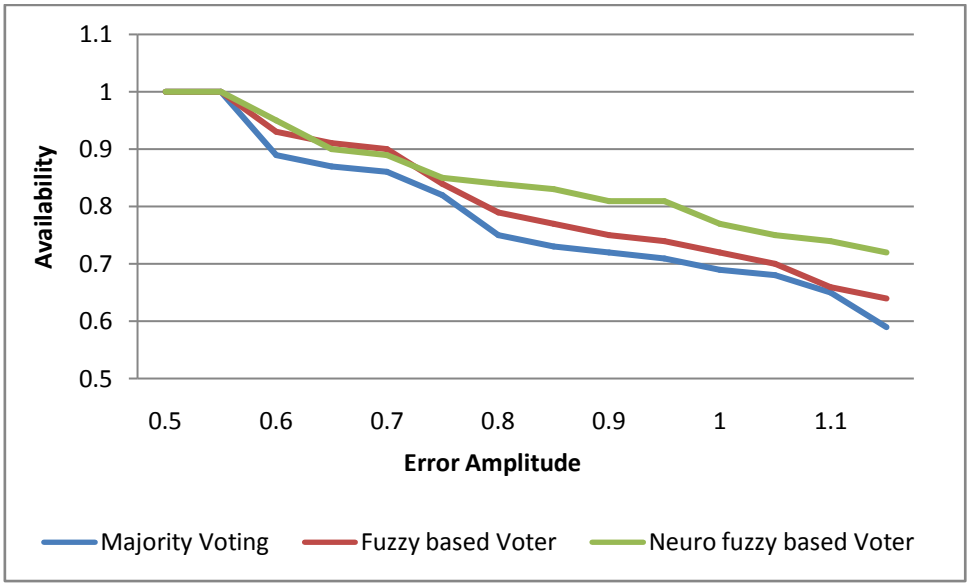

Figure 6: Availability of Voter

The proposed neuro-fuzzy voter achieves improved availability compared to the standard majority and fuzzy voters. When compared to majority voter, the proposed neuro-fuzzy method has higher availability by 3.39% to 19.85%. The proposed neuro-fuzzy method improves availability by 1.11% to 11.76% when compared to fuzzy voter.

Conclusion

This paper presented a neuro-fuzzy voting scheme design for fault tolerant systems encompassing fuzzy reasoning and fuzzy control, where some IF–THEN rules were initially learned with training data and/or expert knowledge. Rules are later be generated (interpolated) for varied input conditions. The proposed neuro-fuzzy voter's performance was tested on a refined experimental harness which permitted modeling of various distributions of input signal's noise and errors. The plots show that the new neuro-fuzzy voter significantly improved regarding safety and availability measures. When reasonable noise amplitudes are present, neuro-fuzzy voting schemes provides greatly improved voting performance.

Reference

[1]. Nelson, V. P. (1990). Fault-tolerant computing: Fundamental concepts. Computer, 23(7), 19-25.

[2]. Joshi, B., Pradhan, D., &Stiffler, J. (2008). Fault-Tolerant Computing. Wiley Encyclopedia of Computer Science and Engineering.

[3]. Avizienis, A. (1997). Toward systematic design of fault-tolerant systems. Computer, 30(4), 51-58.
[4]. Huang, Y., &Kintala, C. (1995). Software fault tolerance in the application layer. Software Fault Tolerance, 3, 231-248.
[5]. Zhang, X., Wang, E., Tang, F., Yang, M., Wei, H., & Dong, X. (2013). A dual process redundancy approach to transient fault tolerance for ccNUMA architecture. Neurocomputing, 122, 50-57.
[6]. M.H. Azadmanesh, A.W. Krings, "Asynchronous Behavior of Egoestic Voting Algorithms", Proc. 5th World Multi-Conference on Systemics, Cybernetics and Informatics, SCI 2001, July 22-25, 2001, Orlando, Florida USA
[7]. Kim, M. H., Lee, S., & Lee, K. C. (2010). Kalman predictive redundancy system for fault tolerance of safety-critical systems. Industrial Informatics, IEEE Transactions on, 6(1), 46-53.
[8]. Lorczak, P. R. Caglayan A. K. and Eckhardt. D. E. A Theoretical Investigation of Generalised Voters", Proc. of IEEE 19th Ann. Int. Symp. on Fault-Tolerant Computing Systems, 1989, Chicago, USA, June, pp. 444-451.
[9]. Latif-Shabgahi, G., Bass, J. M., & Bennett, S. (2004). A taxonomy for software voting algorithms used in safety-critical systems. Reliability, IEEE Transactions on, 53(3), 319-328.
[10]. Siewiorek, D. P. and R. S. Swarz, Reliable Computer Systems: Design and Evaluation, Digital Press, 1992.
[11]. Cai, X., Lyu, M. R., &Vouk, M. A. (2005, November). An experimental evaluation on reliability features of N-version programming. In Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on (pp. 10-pp). IEEE.
[12]. Latif-Shabgahi, G., & Hirst, A. J. (2005). A fuzzy voting scheme for hardware and software fault tolerant systems. Fuzzy Sets and Systems, 150(3), 579-598.
[13]. K. Kim, M.A. Vouk, D.F. McAllister, Fault tolerant software voters based on fuzzy equivalence relations, Proc. IEEE Aerospace Conf. 4 (1998) 5–19.
[14]. Karimi, A., Zarafshan, F., Jantan, A., Ramli, A. R., Saripan, M., & Al-Haddad, S. A. R. (2012). Exact Parallel Plurality Voting Algorithm for Totally Ordered Object Space Fault-Tolerant Systems. Pertanika Journal of Science & Technology, 20(1).
[15]. Singamsetty, P., & Panchumarthy, S. (2011). A novel history based weighted voting algorithm for safety critical systems. Journal of Advances in Information Technology, 2(3), 139-145.
[16]. Zarafshan, F., Latif-Shabgahi, G. R., & Karimi, A. (2010, July). A novel weighted voting algorithm based on neural networks for fault-tolerant systems. In Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on (Vol. 9, pp. 135-139). IEEE.

[17]. Kwiat, K., Taylor, A., Zwicker, W., Hill, D., Wetzonis, S., &Ren, S. (2010, November). Analysis of binary voting algorithms for use in fault-tolerant and secure computing. In Computer Engineering and Systems (ICCES), 2010 International Conference on (pp. 269-273). IEEE.

[18]. Linda, O., & Manic, M. (2011). Interval type-2 fuzzy voter design for fault tolerant systems. Information Sciences, 181(14), 2933-2950.

[19]. Latif-Shabgahi, G., Bennett, S., & Bass, J. M. (2003). Smoothing voter: a novel voting algorithm for handling multiple errors in fault-tolerant control systems. Microprocessors and Microsystems, 27(7), 303-313.