Int. J. Advanced Networking and Applications
Volume: 5 Issue: 6 Pages: 2096-2103 (2014) ISSN : 0975-0290

Soft Computing Approaches To Fault Tolerant Systems

[1]Neeraj Prakash Srivastava, 2 Dr R. K. Srivastava
1Research Scholar, Mewar University
2 Associate Professor, Dept of Computer Science, Bareilly College Bareilly, UP, India
Email : rk_srivastava17@rediffmail.com

----------ABSTRACT----------
We present in this paper as an introduction to soft computing techniques for fault tolerant systems and the terminology with different ways of achieving fault tolerance. The paper focuses on the problem of fault tolerance using soft computing techniques. The fundamentals of soft computing approaches and its type with introduction of fault tolerance are discussed. The main objective is to show how to implement soft computing approaches for fault detection, isolation and identification. The paper contains details about soft computing application with an application of wireless sensor network as fault tolerant system.
----

1. Introduction:

Future values of many real world processes having uncertainty are neither exactly governed by a mathematical model nor by probabilistic models. Soft computing, using the relations, a generalization of function, has definitely proved its worth by past researches, to model such situations.

Soft computing provides a computational framework to address, design, analysis and modeling problems in the context of uncertain and imprecise information. Soft computing is tolerant of imprecision, uncertainty, partial truth, and approximation. One of the major thrust areas of research in the field of developing decision system is to provide low cost solutions utilizing the intelligent tools for information processing. The development of hybridized technique like neuro-fuzzy system is one of the fairly applicable tools in the framework of soft computing. In effect, the role model for soft computing is the human mind. The guiding principle of soft computing is to: Exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost and solve the fundamental problem associated with the current technological development: the lack of the required intelligence of the recent information technology that enables human-centered functionality. The basic ideas underlying soft computing in its current incarnation have links to many earlier influences, among them Zadeh's1965 paper on fuzzy sets; the 1975 paper on the analysis of complex systems and decision processes; and the 1979 report (1981 paper) on possibility theory and soft data analysis. The inclusion of neural computing and genetic computing in soft computing came at a later point.

The principal constituents of Soft Computing (SC) are:
• Fuzzy Systems (FS), including Fuzzy Logic (FL);
• Evolutionary Computation (EC), including Genetic Algorithms (GA);
•Neural Networks (NN), including Neural Computing (NC);
• Machine Learning (ML);
• Probabilistic Reasoning (PR).

A fault tolerance is a setup or configurations that prevent a computer or network device from failing in the event of an unexpected problem or error. To make a computer or network fault tolerant requires that the user or company to think how a computer or network device may fail and take steps that help prevent that type of failure. The complementarily of FS, NN, EC, ML and PR has an important consequence:

In many cases a problem can be solved most effectively by using FS, NN, EC, ML and PR in combination rather than exclusively. A striking example of a particularly effective combination is what has come to be known as "neuro fuzzy systems". Such systems are becoming increasingly visible as consumer products ranging from air conditioners and washing machines to photocopiers and camcorders. Less visible but perhaps even more important are neuro fuzzy systems in industrial applications. What is particularly significant is that in both consumer products and industrial systems, the employment of soft computing techniques leads to systems which have high MIQ (Machine Intelligence Quotient). In large measure, it is the high MIQ of SC-based systems that accounts for the rapid growth in the number and variety of applications of soft computing.

1.1 Fuzzy Systems

Fuzzy systems are based on fuzzy logic, a generalization of conventional (Boolean) logic that has been extended to handle the concept of partial truth — truth values between

Volume: 5 Issue: 6 Pages: 2096-2103 (2014) ISSN : 0975-0290

"completely true" and "completely false". It was introduced by L.A. Zadeh of University of California, Berkeley, U.S.A., in the 1960's, as a means to model the uncertainty of natural language. Zadeh himself says that rather than regarding fuzzy theory as a single theory, we should regard the process of "fuzzification" as a methodology to generalize any specific theory from a crisp (discrete) to a continuous (fuzzy) form.

1.1.1 Fuzzy Sets

The theory o f fuzzy sets now encompasses a well organized corpus of basic notions including (and not restricted to) aggregation operations, a generalized theory of relations, specific measures of information content, a calculus of fuzzy numbers. In mathematics fuzzy sets have triggered new research topics in connection with category theory, topology, algebra, analysis. Fuzzy sets are also part of a recent trend in the study of generalized measures and integrals, and are combined with statistical methods. Furthermore, fuzzy sets have strong logical underpinnings in the tradition of many-valued logics. Fuzzy set-based techniques are also an important ingredient in the development of information technologies. In the field of information processing fuzzy sets are important in clustering, data analysis and data fusion, pattern recognition and computer vision.

1.1.2 Fuzzy Logic

The degree to which the statement x is in F is true is determined by finding the ordered pair whose first element is x. The degree of truth of the statement is the second element of the ordered pair. In practice, the terms "membership function" and fuzzy subset get used interchangeably.
We know what a statement like "x is LOW" means in fuzzy logic, how we interpret a statement like: (x is low) AND (y is high) OR (NOT z is medium). The standard definitions in fuzzy logic are:

Truth (NOT x) = 1.0 − truth(x),
Truth (x AND y) = min {truth(x), truth(y)},
Truth (x OR y) = max {truth(x), truth(y)}.

Generally speaking, logic, as a mathematical theory, studies the notions of consequence. It deals with propositions (sentences), sets of propositions and the relation of consequence among them. The task of formal logic is to present all this by means of well-defined logical calculi admitting exact investigation. Various calculi differ in their definitions of sentences and concepts of consequences, e.g. propositional/predicate logics, modal propositional/ predicate logics, many-valued propositional/predicate logics etc.

1.1.3 Fuzzy Numbers and Fuzzy Arithmetic

Fuzzy numbers are fuzzy subsets of the real line. They have a peak or plateau with membership grade 1, over which the members of the universe are completely in the set. The membership function is increasing towards the peak and decreasing away from it. Fuzzy numbers are used very widely in fuzzy control applications. A typical case is the triangular fuzzy number which is one form of the fuzzy number. Slope and trapezoidal functions are also used, as well as exponential curves similar to Gaussian probability densities.

1.2 Neural Networks

There is no universally accepted definition of neural networks (NN), a common characterization says that an NN is a network of many simple processors ("units"), each possibly having a small amount of local memory. The units are connected by communication channels ("connections") which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate only on their local data and on the inputs they receive via the connections. The restriction to local operations is often relaxed during training.
Some NNs are models of biological neural networks and some are not, but historically, much of the inspiration for the field of NNs came from the desire to produce artificial systems capable of sophisticated, perhaps "intelligent", computations similar to those that the human brain routinely performs, and thereby possibly to enhance our understanding of the human brain. NNs normally have great potential for parallelism, since the computations of the components are largely independent of each other. Some people regard massive parallelism and high connectivity to be defining characteristics of NNs, but such requirements rule out various simple models, such as simple linear regression (a minimal feed-forward net with only two units plus bias), which are usefully regarded as special cases of NNs.

Some definitions of neural networks are:

Definition 1:

"A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
1. Knowledge is acquired by the network through a learning process.
2. Interneuron connection strengths known as synaptic weights are used to store the knowledge".

Definition 2:

"A neural network is a circuit composed of a very large number of simple processing elements that are based on neurons. Each element operates only on local information. Furthermore each element operates asynchronously; thus there is no overall system clock".

Definition 3:

"Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store, and utilize experiential knowledge".

Volume: 5 Issue: 6 Pages: 2096-2103 (2014) ISSN : 0975-0290

1.2.1 Principle of Neural Networks

In principle, NNs can compute any computable function, i.e., they can do everything a normal digital computer can do, or perhaps even more.

In practice, NNs are especially useful for classification and function approximation/ mapping problems which are tolerant of some imprecision, which have lots of training data available, but to which hard and fast rules (such as those that might be used in an expert system) cannot easily be applied.

2. Fault Tolerance

2.1 FAULT CHARACTERIZATION

We use fault to mean an abnormal operating condition of the computer system, which affects a running routine in some way. The routine fails only when one or more faults cause it to compute the wrong answer.

We give two classifications of faults. The first is hard and second is soft:

(a).Hard faults: Cause program interruption and are outside the scope of what the executable program can directly detect. These faults can result from hardware failure or from data integrity faults that lead to an in correct execution path. An example of a hard fault would be the operating system crashing, causing the program to stop executing.

(b).Soft faults: Do not cause immediate program interruption and are detectable via introspection by user code. Soft faults occur as incorrect floating point or integer data, or perhaps incorrect address values that still point to valid user data space. The second characterization applies only to soft faults, and describes their temporal behavior:

(c). *Persistent fault:* The incorrect bit pattern will not change as execution proceeds. Example: The primary source of a data value is incorrect, so there is no ability to restore correct state.

(d).Sticky fault: The incorrect bit pattern can be corrected by direct action. Example: A backup source for the data exists and can be used to restore correct state.

(e).Transient fault: The incorrect pattern occurs temporarily. Example: Data in a cache is incorrect, but the correct value is still present in main memory and the cache value is flushed.

2.2 REQUIREMENTS

The requirements of fault tolerance are as:

a) *Dependable Systems:* Hazards to systems are a fact of life. So are faults. Yet we want our systems to be dependable. A system is dependable when it is trustworthy enough that reliance can be placed on the service that it delivers. For a system to be dependable, it must be available, reliable, safe and secure. Although these system attributes can be considered in isolation, in fact they are interdependent. For instance, a system that is not reliable is also not available. A secure system that doesn't allow an authorized access is also not available. An unreliable system to control nuclear reactors is probably not a safe one either.

b) *Approaches to Achieving Dependability:* Achieving the goal of dependability requires effort at all phases of a system's development. Steps must be taken at design time, implementation time, and execution time, as well as during maintenance and enhancement. At design time, we can increase the dependability of a system through fault avoidance techniques. At implementation time, we can increase the dependability of the system through fault removal techniques. At execution time, fault tolerance and fault evasion techniques are required.

2.3 STRATAGIES TO HANDEL FAULTS

a) *Fault Avoidance:* Fault avoidance uses various tools and techniques to design the system in such a manner that the introduction of faults is minimized. A fault avoided is one that does not have to be dealt with at a later time. Techniques used include design methodologies, verification and validation methodologies, modeling, and code inspections and walk-through.

b) *Fault Removal:* Fault removal uses verification and testing techniques to locate faults enabling the necessary changes to be made to the system. The range of techniques used for fault removal includes unit testing, integration testing, regression testing, and back-to-back testing. It is generally much more expensive to remove a fault than to avoid a fault.

c) *Fault Tolerance:* In spite of the best efforts to avoid or remove them, there are bound to be faults in any operational system. A system built with fault tolerance capabilities will manage to keep operating, perhaps at a degraded level, in the presence of these faults. For a system to be fault tolerant, it must be able to detect, diagnose, confine, mask, compensate and recover from faults.

d) *Fault Evasion:* It is possible to observe the behavior of a system and use this information to take action to compensate for faults before they occur. Often, systems exhibit a characteristic or normal behavior. When a system deviates from its normal behavior, even if the behavior continues to meet system specifications, it may be appropriate to reconfigure the system to reduce the stress on a component with a high failure potential. We have coined the term fault evasion to describe this practice. For example, a bridge that sways as traffic crosses may not be exceeding specifications, but would warrant increased attention from a bridge inspector. Similarly, a computer system that suddenly begins to respond sluggishly often prompts a prudent user

Volume: 5 Issue: 6 Pages: 2096-2103 (2014) ISSN : 0975-0290

to backup any work in progress, even though overall system performance may be within specification.

2.4 FAULT CLASSES

No system can be made to tolerate all possible faults, so it is essential that the faults be considered throughout the requirements definition and system design process. However, it is impractical to enumerate all of the faults to be tolerated; faults must be aggregated into manageable fault classes. Faults may be classified based on Locality (atomic component, composite component, system, operator, environment), on Effect (timing, data), or on Cause (design, damage). Other possible classification criteria include Duration (transient, persistent) and Effect on System State (crash, amnesia, partial amnesia, etc.).

Since the location of a fault is so important, fault location is a logical starting point for classifying faults.

1) Locality

(i).Atomic Component Faults:

Concept Definition**:** An atomic component fault is a fault at the fault floor, that is, in a component that cannot be subdivided for analysis purposes.

Bridge Example**,** A fault in an individual structural member in a bridge may be considered a atomic component fault. If the bridge design properly distributes the load among the various structural members (resources) of the bridge, then the load is transferred to other structural members, no failure occurs, and the fault is masked. The fault may be detected by observation of cracks or deformation, or it may remain latent. Computer System Example, in a computer system, substrate faults can appear in diverse forms. For instance, a fault in a memory bit is not an atomic component fault if the details of the memory are below the current span of concern. Such a fault may or may not appear as a memory fault, depending upon the memory's ability to mask bit faults.

(ii). Composite Component Faults:

Concept Definition: A composite component fault is one that arises within an aggregation of atomic components rather than in an atomic component. It may be the result of one or more atomic component faults. Bridge Example, A pier failure would be an example of a composite component failure for a bridge. Computer System Example**,** A disk drive failure in a computer system is an example of a composite component failure. If the individual bits of memory are considered to be in the span of concern, a failure of one of those would be a component failure as well.

(iii). System Level Faults:

Concept Definition: A system level fault is one that arises in the structure of a system rather than in the system's components. Such faults are usually interaction or integration faults, that is, they occur because of the way the system is assembled rather than because of the integrity of any individual component. Note that an inconsistency in the operating rules for a system may lead to a system level fault. System level faults also include operator faults, in which an operator does not correctly perform his or her role in system operation. Systems that distribute objects or information are prone to a special kind of system fault: replication faults. Replication faults occur when replicated information in a system becomes inconsistent, either because replicates that are supposed to provide identical results no longer do so, or because the aggregate of the data from the various replicates is no longer consistent with system specifications. Replication faults can be caused by malicious faults, in which components such as processors "lie" by providing conflicting versions of the same information to other components in the system. Malicious faults are sometimes called Byzantine faults after an early formulation of the problem in terms of Byzantine generals trying to reach a consensus on attacking when one of the generals is a traitor. Bridge Example, A bridge failure resulting from insufficient allowance for thermal expansion in the overall structure could be considered a system failure: individual structural members behave as specified, but faulty assembly causes failures when they interact. Computer System Example, Consider the computer systems in an automobile. Suppose the airbag deployment computer and the anti-lock brake computer are both known to work properly and yet fail in operation because one computer interferes with the other when they are both present. This would be a system fault.

(iv)External Faults: External faults arise from outside the system boundary, the environment, or the user. Environmental faults include phenomena that directly affect the operation of the system, such as temperature, vibration, or nuclear or electromagnetic radiation or that affects the inputs provided to the system. User faults are created by the user in employing the system. Note that the roles of user and operator are considered separately; the user is considered to be external to the system while the operator is considered to be a part of the system.

(v).Effects: Faults may also be classified according to their effect on the user of the system or service. Since computer system components interact by exchanging data values in a specified time and/or sequence, fault effects can be cleanly separated into timing faults and

Volume: 5 Issue: 6 Pages: 2096-2103 (2014) ISSN : 0975-0290

value faults. Timing faults occur when a value is delivered before or after the specified time. Value faults occur when the data differs in value from the specification.

(a).Value Faults: Computer systems communicate by providing values. A value fault occurs when a computation returns a result that does not meet the system's specification. Value faults are usually detected using knowledge of the allowable values of the data, possibly determined at run time.

(b).Timing Faults: A timing fault occurs when a process or service is not delivered or completed within the specified time interval. Timing faults cannot occur if there is no explicit or implicit specification of a deadline. Timing faults can be detected by observing the time at which a required interaction takes place; no knowledge of the data involved is usually needed. Since time increases monotonically, it is possible to further classify timing faults into early, late, or "never" (omission) faults. Since it is practically impossible to determine if "never" occurs, omission faults are really late timing faults that exceed an arbitrary limit. Systems that never produce value faults, but only fail by omission are called fail-silent systems. If all failures require system restart, the system is a fail-stop system.

(c).Duration: Persistent faults remain active for a significant period of time. These faults are sometimes termed hard faults. Persistent faults usually are the easiest to detect and diagnose, but may be difficult to contain and mask unless redundant hardware is available. Persistent faults can be effectively detected by test routines that are interleaved with normal processing. Transient faults remain active for a short period of time. A transient fault that becomes active periodically is a periodic fault (sometimes referred to as an intermittent fault). Because of their short duration, transient faults are often detected through the faults that result from their propagation.

(d).Immediate Cause: Faults can be classified according to the operational condition that causes them. These include resource depletion, logic faults, or physical faults. Resource depletion faults occur when a portion of the system is unable to obtain the resources required to perform its task. Resources may include time on a processing or communications device, storage, power, logical structures such as a data structure, or a physical item such as a processor. Logic faults occur when adequate resources are available, but the system does not behave according to specification. Logic faults may be the result of improper design or implementation, as discussed in the next section. Logic faults may occur in hardware or software. Physical faults occur when hardware breaks or a mutation occurs in executable software. Most common fault tolerance mechanisms deal with hardware faults.

(e).Ultimate Cause: Faults can also be classified as to their ultimate cause. Ultimate causes are the things that must be fixed to eliminate a fault. These faults occur during the development process and are most effectively dealt with using fault avoidance and fault removal techniques. A common ultimate cause of a fault is an improper requirements specification which leads to a specification fault. Technically this is not a fault, since a fault is defined to be the failure of a component/interacting systems and a failure is the deviation of the system from specification. However, it can be the reason a system deviates from the behavior expected by the user. An especially insidious instance of this arises when the requirements ignore aspects of the environment in which the system operates. For instance, radiation causing a bit to flip in a memory location would be a value fault which would be considered an external fault. However, if the fault propagates inside the system boundary the ultimate cause is a specification fault because the system specification did not foresee the problem. Flowing down the waterfall, a design fault results when the system design does not correctly match the requirements, and an implementation fault arises when the system implementation does not adequately implement the design. The validation process is specifically designed to detect these faults. Finally, a documentation fault occurs when the documented system does not match the real system.

2.5 FAULT TOLERANCE IN GENERAL PURPOSE COMPUTER

People may not realize to which extent fault-tolerance techniques are used in general-purpose computers to increase their reliability. Techniques used in general-purpose computers are also utilized in more specialized fault-tolerant computers, so it is a good starting point to study these computers. Based on the assumption that most errors are transient, recovery consists primarily of retry by the error detection mechanisms. A retry is usually not done immediately, but after a pause. During that time, the source of the transient error, e.g. power instability, might have disappeared. A computer is usually divided into three main sections: processor, primary memory and I/O. These sections often employ slightly different fault-tolerant techniques. In the more expensive computers, and now also increasingly on cheaper computers, double-error-detecting codes are also used. In addition, parity is used on address and control information. Recovery can be done with single error-correcting codes on data and retry on address and control information parity error. Memory, under software control, can in some systems be dynamically reconfigured to exclude bad pages.

Many of the techniques used on memory, can also be used on I/O. Retry is often extensively used here, especially on devices as disks this is an effective approach. A processor

Volume: 5 Issue: 6 Pages: 2096-2103 (2014) ISSN : 0975-0290

contains many registers. To provide fault-tolerance here, the same techniques as those used on memory can be used. In addition, duplication of control logic is commonly used. To increase availability, repair time has to be minimized. One way to do this is remote diagnostics. When a fault is detected, either the computer or an operator notifies a service center, possibly located far Detection Recovery Memory Parity and double-error-detecting code Single error-correction code, retry and dynamically reconfigurable memory I/O Parity Retry Processor Parity, duplication and comparison retry away from the computer site. The service center can connect to the computer, and use diagnostic programs if necessary. The personnel at the service center can either fix the problem from their site or ship a replacement module to the failing site.

2.6 FAULT TOLERANCE MECHANISM

(i).Characteristics Unique to Digital Computer Systems: Digital computer systems have special characteristics that determine how these systems fail and what fault tolerance mechanisms are appropriate. First, digital systems are discrete systems. Unlike continuous systems, such as analog control systems, they operate in discontinuous steps. Second, digital systems encode information. Unlike continuous systems, values are represented by a series of encoded symbols. Third, digital systems can modify their behavior based on the information they process.

(i).Redundancy Management: Fault tolerance is sometimes called redundancy management. Redundancy is necessary, but not sufficient for fault tolerance. For example, a computer system may provide redundant functions or outputs such that at least one result is correct in the presence of a fault, but if the user must somehow examine the results and select the correct one, and then the only fault tolerance is being performed by the user. However, if the computer system correctly selects the correct redundant result for the user, then the computer system is not only redundant, but also fault tolerant. Redundancy management marshals the no-fault resources to provide the correct result. Redundancy management or fault tolerance involves the following actions. Fault Detection: The process of determining that a fault has occurred. Fault Diagnosis The process of determining what caused the fault, or exactly which subsystem or component is faulty.

(iii) Fault Containment: The process that prevents the propagation of faults from their origin at one point in a system to a point where it can have an effect on the service to the user.

Fault Masking: The process of insuring that only correct values get passed to the system boundary in spite of a failed component.

Fault Compensation: If a fault occurs and is confined to a subsystem, it may be necessary for the system to provide a response to compensate for output of the faulty subsystem.

Fault Repair: The process in which faults are removed from a system. In well designed fault tolerant systems, faults are contained before they propagate to the extent that the delivery of system service is affected.

3. Soft Computing approach

Artificial Neural network based methods often requires preprocessing algorithm to reduce the effect of noise, distortions and to increase the fault occurrences. Another technique has been combined neural network including fuzzy logic and genetic algorithm.

The study shows that potential for soft computations to increase fault tolerance in general purpose CPU. There are three important characteristic of soft computing that make them resilient of error that is first, redundancy, adaptivity and reduce precision. Second, the extent to which soft computing is fault tolerant on general purpose CPU by conducting fault injection experiments and do not alter the numerical results of the computations. Third, the development of light weight recovery technique that tries to check point and recover only “Hard State”. Soft computations as compare to traditional numerically oriented computations shows increased resilience to fault because soft computing permit a less strict definition of program correctness due to qualitative nature of their results and output state is numerically correct with in some tolerance as well as quality wise correct based on higher level interpretation.

An important work load produced results having higher user level interpretation, such computation as soft computation. The data corruptions can change the numerical result of soft computing. System that can identify and exploit such error resiliency at the user level offer new opportunities for fault tolerance optimizations. The researchers have observed that computing characteristics and proposed to exploit them for reduced energy consumptions as well as for fault tolerances.

4. Fault Tolerance in Wireless Sensor Networks

Applications such as security and surveillance monitoring, battlefield command and control, and wildlife or medical monitoring rely on the correct functioning of the underlying WSNs for data sensing and retrieval in response to

Volume: 5 Issue: 6 Pages: 2096-2103 (2014) ISSN : 0975-0290

application queries. In such applications, the WSNs are often deployed in an area where replacements of sensors are difficult or impossible. We consider three major sources of faults that could cause a WSN to fail. One source is due to energy depletion of Sensor Nodes, such that the underlying WSN simply exhausts its energy to be able to answer queries. Another source is due to sensor faults including measurement faults. The third source is due to communication faults because of noise and interference in the WSN. To conserve energy of SNs, a well accepted approach is for the WSN to self-organize itself into clusters. Within a cluster, a cluster head is elected to perform more data aggregation and relay duties than normal SNs and is rotated applications concerned with sensor readings such as the minimum/maximum/average of sensor data, cluster heads can also perform training part of back propagation functions to reduce error. A cluster can be Distributed manner on a per-sensor-node basis. To cope with the second source of faults, i.e., sensor faults, a general approach is to incorporate redundancy to allow sensor faults to be detected, isolated, and corrected so that the system can continue to function correctly in data sensing and retrieval. However, the use of redundancy impacts the energy consumption rate of the system since more SNs would need to be used as redundancy to achieve sensor fault tolerance. Therefore, there is a tradeoff between these two sources of faults. On the one hand, we like to incorporate redundancy to deal with sensor faults. On the other hand, redundancy should be used only as needed so as not to quickly deplete the energy of the system. Current research work on fault tolerance mechanisms to cope with sensor faults in WSNs can be classified into hardware redundancy, time redundancy and information redundancy. Hardware redundancy utilizes extra hardware for fault detection or masking. For example, two temperature sensors can be used to agree on a temperature reading before the reading is considered as a correct response. If a discrepancy exists, then a third temperature sensor can be used to reduce the error and find the optimize value of third temperature sensor. A sensor can also be made to disambiguate a sensor measurement fault from a true event by using back propagation algorithm after comparing readings obtained from its neighbor sensors of the same type. The time to live (TTL) value of query and reply packets is adjusted to allow multiple readings to return to the processing center through multiple paths. Time redundancy is a simple form of fault tolerance that utilizes repeated execution as the primary mechanism. One can monitor the output of a sensor reading query to see if the output returned is within a normal range. If the reading is out of ordinary, a second reading query can be performed with the output. Information redundancy uses the relationship among sensor data from the physical world for fault detection. For example, a relation exists between speed, pressure and position in a diesel engine such that if the pressure sensor is detected to be faulty, one can deduce its value from the other two sensor readings.

We consider a WSN as having experienced a failure when it fails to deliver sensor data correctly in response to an application-level query, due to one of the three sources of faults, i.e., energy depletion, sensor fault, or communication fault.

5. CONCLUSION

Fault-tolerance is achieved by applying a set of analysis and design techniques to create systems with dramatically improved dependability. In this we discussed different type of faults and fault tolerance in conclusion; we got all the information about fault characteristics. As new technologies are developed and new applications arise, new fault-tolerance approaches are also needed. In the early days of fault-tolerant computing, it was possible to craft specific hardware and software solutions from the ground up, but now chips contain complex, highly-integrated

Functions, and hardware and software must be crafted to meet a variety of standards to be economically viable

REFERENCES:

[1]. Aggarwal R K, Xuan Q Y, Johns A T, Li F R, Bennett A, (1999), A novel approach to fault diagnosis in multi-circuit transmission lines using fuzzy ARTmap neural networks, IEEE transactions on neural networks, Vol.10, No.5,pp.1214-1221.

[2]. Altug S, Chow MY, Trussell HJ, (1999), Fuzzy inference systems implemented on neural architectures for motor fault detection and diagnosis, IEEE transactions on industrial electronics, 1999, Vol.46, No.6, pp.1069-1079.

[3]. Aminian M, Aminian F, (2000), Neural-network based analog-circuit fault diagnosis using the wavelet transform as pre-processor, IEEE transactions on circuits and systems ii-analog and digital signal processing Vol.47, No.2, pp.151-156.

[4]. Brown M & Harris C J, (1994), Neuro-fuzzy adaptive modelling and control, Prentice Hall. Brown M & Harris C J, (1994b), The Modelling Abilities of the Binary CMAC, IEEE Int.Conf. Neural Networks,pp 1335-133

[5]. Chen J, Patton R J (1999), Robust Model Based Fault Diagnosis For Dynamic Systems, Kluwer Academic Publishers ISBN 0-7923-8411-3.

[6]. Chen J, Patton, R J & G P Liu, (1997), Robust fault detection of dynamic systems via genetic algorithms, *Pooc. Instn. Mech Engrs,* 211, Part I, pp357-364.

[7]. Cristian, F., "Understanding Fault-Tolerant Distributed Systems.", Communications of the ACM, vol. 34 no. 2, Feb 1991, 56-78.

[8]. Crowther W J, Edge K A, Burrows C R, Atkinson R M & Woollons D J, (1998), Fault diagnosis of a hydraulic actuator circuit using neural networks an output vector space classification approach *Journal of Systems & Control Engineering*, 212, (1), pp57-68.

Volume: 5 Issue: 6 Pages: 2096-2103 (2014) ISSN : 0975-0290

[9]. Dalmi I, Kocvacs L, Lorant I & Terstyansky G, (1999), Application of Supervised and Unsupervised Learning Methods to Fault Diagnosis, *Proc. 14th World*

[10]. Dong D & McAvoy T J, (1996), Non-linear Principal Component Analysis - Based on Principal Curves and Neural Networks, Computers and Chemical Engineering 20, pp. 65-78.

[11]. Farag W A, Quintana V H & Lambert-Torres G (1998), A Genetic-Based Neuro-Fuzzy Approach for Modelling and Control of Dynamical Systems, *IEEE Trans. Neural Networks*, 9, (5).

[12]. J. Korbicz, J.M. Koscielny, Z. Kowalczuk, and W. Cholewa (eds.), Fault Diagnosis. Models, Artificial Intelligence, Apllications. Berlin: Springer-Verlag, 2004.

[13]. Kozyrakis, Christoforos E., and David Patterson, *A New Direction for Computer Architecture Research*, Computer, Vol. 31, No. 11, November 1998.

[14]. Laprie, J. C. (ed.), Dependability: Basic Concepts and Terminology, Vienna, Springer-Verlag, 1992.

[15]. Melliar-Smith 91, Melliar-Smith, P. M. "A Project to Investigate Data-base Reliability",Report, Computing Lab., University of Newcastle-upon-Tyne, England, 1975.

[16]. Naidu S, Zafirou E & McAvoy T J (1990), Use of Neural-networks for failure detection in a control system, *IEEE Control Sys. Magazine*, 10, 49-55.

[17]. Patton R J, (1997), Robustness in model-based fault diagnosis: The 1997 Situation, A Rev. Control,21, 103-123, Pergamon Press.

[18]. Narendra K S & Parthasarathy K, (1990),Identification and control of dynamic systems using neural networks, *IEEE Trans. on Neural Network*, 1, 4-27.

[19]. Narendra K S, (1996), Neural Networks for Control: Theory and Practice, Proc of IEEE, Oct, 84 (10),1385-1406. 84 (10), 1385-1406.

[20]. Narendra K G, Sood V K, Khorasani K, Patel R, (1998), RBF Neural Network for fault diagnosis in a HVDC system, *IEEE transactions on power systems*, Vol.13, No.1, pp.177- 183.

[21]. Obuchowicz & Korbicz J, (1998), Evolutionary Search with soft selection and forced direction of mutation", Proceedings of 7th Int. Symp. *Intelligent Information System*, Malbork, Poland, June 15-19, pp300-309.

[22]. Pantelelis N G, Kanarachos A E, Gotzias N, (2000), Neural networks and simple models for the fault diagnosis of naval turbochargers, *Mathematics and computers in simulation* Vol.51, No.3-4, pp.387-397.

[23]. Patton R J, Chen J & Liu G P, (1997), Robust fault detection of dynamic systems via genetic algorithms, *Proc. of IMechE Part I-J. of Syst. &Contr. Eng*. 211(5): 357-364.

[24]. Patton R J, Chen J & Lopez-Toribio C J, (1998), Fuzzy observers for non-linear dynamic systems fault diagnosis. *Proc. 37th IEEE Conf. On Decision and Control,* pp84-89.

[25]. Patton R J, Frank P M & Clark R N, (2000), Issues in fault diagnosis for dynamic systems, Springer-Verlag, London, April 2000.

[26]. Patton R J, Lopez-Toribio C J & Uppal F J, (1999), Artificial Intelligence Approaches to Fault Diagnosis, *Applied Mathematics and Computer Science*, Technical University of Zielona Gora, Poland, Vol. 9, No. 3, 471-518.

[27]. Pfeufer T, Ayoubi M, (1997), Application of a hybrid neuro-fuzzy system to the fault diagnosis of an automotive electromechanical actuator, *Fuzzy sets and systems*, Vol.89, No.3, pp.351-360.

[28]. Ren X & Chen J, (1999), A Modified Neural Network For Dynamical System Identification & Control, *Proc. 14th World Congress of IFAC* ISBN 008 043248 4.

[29]. R.J.Patton, P.M. Frank, and R.N. Clark(eds.), Issues of Fault Diagnosis for Dynamic Systems, Berlin: Springer-Verlag,2000.

[30]. Schneider H & Frank P M, (1994), Fuzzy logic based threshold adaptation for fault detection in robots, *Proc. of The Third IEEE Conf. on Control Applications*, Glasgow, Scotland, pp-1127-1132.

[31]. Schneider H & Frank P M, (1996), Observer-based supervision and fault detection in robots using non-linear and fuzzy-logic residual evaluation, *IEEE Trans. Contr. Sys. Techno*. 4, (3), pp274-282

[32]. Sharif M A & Grosvenor R I, (1998), Process plant condition monitoring and fault diagnosis, *Journal of Process Mechanical Engineering*, 212, (1), pp13-30.Shen Q Leitch R, (1993), Fuzzy Qualitative Simulation, *IEEE Trans. Sys. Man & Cybernetics*, SMC-23, (4), pp1038-1061.

[33]. Soliman A, Rizzoni G, Kim YW, (1999), Diagnosis of an automotive emission control system using fuzzy inference, *Control engineering practice*,