System Diagnosis and Fault Tolerance for Distributed Computing System: A Review

1. Nilotpal Baruah,
PhD Research Scholar, Dept. of Comp. Sc., Assam University, Silchar, India
nilotpaldu@gmail.com

2. Dr. Lakshmi P. Saikia,
Professor, Dept. of Computer Sc.& Engg., Assam down town University, Guwahati, India
lp_saikia@yahoo.co.in

3. Dr. K. Hemachandran,
Professor, Dept. of Comp. Sc., Assam University, Silchar, India
khchandran@rediffmail.com

Abstract

An adaptive system diagnosis fault tolerance method for distributed system. The system is comprised of a network including N nodes where N is integer and greater than equal to 3 and each node is able to execute an algorithm to communicate with the network. A computer network, often simply referred to as a network, is a collection of hardware components and computers interconnected by communication channels that allow sharing of resources and information. As computer network is a collection of hardware components it is very often that is may have some fault either in the hardware or in the software of the entire network. So to deal with these kinds of faults either hardware of software, some fault diagnosis and fault tolerance mechanism to be implemented for the proper functioning of the system. For such a fault detection and fault tolerant mechanism is to be discussed in this paper. What kind of fault and how they occur will discuss and try to find out some suitable solution of our proposed problem. Various fault detecting mechanism and fault tolerant methodology to be study here and the main goal of the study is to find out some automatic fault detection and fault tolerance techniques.

Keywords
Distributed System, Network, Fault Tolerance, System Diagnosis, LAN, Middleware, Topology, Client –Server, DCS, Simulation.

1. Introduction

Distributed computing is a method of computer processing in which a program is subdivided and different parts are run simultaneously on more than one computer and they are connected over a network. A distributed system is a collection of multiple autonomous computers that interact with each other in order to achieve a common goal over a network. Distributed program computer program that runs in a distributed system, and distributed programming is the process of writing such programs. Distributed computing also refers to the use of distributed systems to solve computational problems. In distributed computing, a problem is divided into many tasks, each of which is solved by one or more computers. Message passing is the technique in distributed computing which is used to communicate with each other.

2. Fault and Fault Tolerance in Distributed Systems

An ideal system is one which is perfectly reliable and never fails, which is impossible to achieve in practice. Systems fail for many reasons. A system can fail at several levels.

2.1. Faults, Errors and Failures

a) Fault is a defect within the system
b) Error is observed by a deviation from the expected behavior of the system
c) Failure occurs when the system can no longer perform as required (does not meet spec)

Failure in a distributed system = when a service cannot be fully provided

i. System failure may be partial
ii. A single failure may affect other parts of a system (failure escalation)

2.2. Fault Tolerance

Fault Tolerance is ability of system to provide a service, even in the presence of errors. Fault tolerance in distributed systems is achieved by**:**

a) **Hardware redundancy**, i.e. replicated facilities to provide a high degree of availability and fault tolerance
b) **Software recovery,** e.g. by rollback to recover systems back to a recent consistent state upon detection of a fault

2.3. Failure Models in Distributed Systems

Failure Types in Server:

a) Crash – server halts, but was working ok until then, e.g. O.S. failure
b) Omission – server fails to receive or respond or reply, e.g. server not listening or buffer overflow
c) Timing – server response time is outside its specification, client may give up
d) Response – incorrect response or incorrect processing due to control flow out of synchronization
e) Arbitrary value (or Byzantine) – server behaving erratically, for example providing arbitrary responses at arbitrary times. Server output is inappropriate but it is not easy to determine this to be incorrect. E.g. duplicated message due to buffering problem. Alternatively there may be a malicious element involved.

The purpose of fault tolerance is to increase the dependability of a system. A complementary but separate approach to increasing dependability is fault prevention. This consists of techniques, such as inspection, whose intent is to eliminate the circumstances by which faults arise. Multiple embedded controllers are used in a distributed control system which communicates with each other. These embedded controllers are devised to control, monitor or assist the operation of equipment, machinery or plant in a distributed system environment.

3. Background

In the past decade, distributed systems have been rapidly evolved, from simple client/server applications in local area networks, to large-scale cloud platforms and Internet-scale P2P systems deployed on tens of thousands of nodes across administrative domains and geographical areas. Despite of the growing popularity and interests, designing and implementing these systems remains challenging, due to their ever-increasing scales as well as the complexity and unpredictability of the system executions.

Various techniques have been proposed for detecting faults in distributed systems. Based on how the expected behavior of a target system is designed, one may check the system against specified properties.

4. Review of Literature

Over the recent decades, there has been a continuous effort to enhance the understanding of the distributed systems for fault tolerance.

From the late sixties considerable amount of work has been carried out on fault tolerance in distributed system. Preparata et. al. had introduced the concept of System level fault diagnosis in 1967, as a technique aimed at diagnosing faults in system composed by a number of units interconnected by a wired point-to point network based on the outcomes of reciprocal tests performed by the units themselves. In the original model by Preparata et. al., a test involves a pair of adjacent (i.e., directly connected) units: the *testing* and the *tested* unit. The testing unit sends a test task to the tested unit, which, in turn, computes the result and returns it to the testing unit. The testing unit generates the test outcome by comparing the returned result with the expected one: if they agree the outcome is 0, otherwise it is 1. Different authors have reviewed the concept of fault tolerance computing systems, like Ramamoorthy[1967], Short[1968], Avizienis[1971], Khul and Reddy[1980,1981], Bagchi and Hakimi[1991].

Hayes[1976] has taken graph theory as the base for his approach towards fault-tolerance design of computing systems.

Lamport Leslie[1984] has described a general method for implementing any desired form of synchronization in a distributed system with any desired degree of fault-tolerance. The synchronization in specified in terms of a state machine and Algorithm is used to execute that state machine. The fault-tolerance of the Algorithm is obtained by the use of a solution to the Byzantine Generals problem. Algorithm is conceptually simplified by considering each process to issue a command on every clock tick, and executing all these commands in sequence. By using synchronized clocks that read absolute time, the algorithm can be implemented in such a way that most of the message transmissions and state machine executions are performed by doing nothing. In practice, each process executes an interrupt-driven program. There, the state machine worked in "logical time", and an explicit

response from every other process was needed to advance a process's logical clock. Failure of any process prevented further progress, so no fault-tolerance was obtained. With Algorithm III, the operation of the state machine occurs in real clock time, which advances regardless of the actions of any other process. Hence, process failure does not impede system progress. The use of clocks allows the elimination of acknowledgment messages. Instead of waiting for a response, the normal situation can become waiting for a no response. However, this may increase the delay because when waiting for a no response, a process always has to wait for the worst-case response time. Acknowledgment messages can also be eliminated by using timeout, but Algorithm gives smaller delays when good clock synchronization mechanisms are employed.

Holt and Smith[1985] have proposed a general diagnosis model for distributed systems and presented the computer system diagnosis that is performed by a system itself, rather than by an outside mechanism. The devices performing the diagnosis and the devices communicating diagnostic information are included in the system model.

Nancy et. al.[1986] have studied VAXClusteres on closely coupled distributed systems. To achieve performance in a multi computer environment, anew communications architecture, communications hardware, and distributed software were jointly designed. The software is a distributed version of the VAX/VMS operating system that uses a distributed lock manager to synchronize access to shared resources. The communications hardware includes a 70 megabit per second message-oriented interconnect and an interconnect port that performs communications tasks traditionally handled by software. A principal goal in their study of VAXclusters was the development of an available and extensible multicomputer system built from standard processors and a general purpose operating system. Much was gained by the joint design of distributed software, communications protocols, and hardware aimed to meet this goal. For example:

a) The CI inter connect supports fast message transfer needed by system software.
b) The CI port implements many of the functions needed by SCA software.
c) The HSC-50 with its message protocol and request queuing optimization logic provides direct network access to a large pool of disks for multiple hosts.

Designing hardware and software together allows for system-level tradeoffs; the software interface and protocols can be tuned to the hardware devices. An important simplifying aspect of the VAXcluster design is the use of a distributed lock manager for resource synchronization. In this way, higher level services such as the file system do not require special code to handle sharing in a distributed environment. However, performance of the lock manager becomes a crucial factor. Distributed lock manager performance has been attacked with the design of a locking protocol requiring a fixed number of messages, independent of the number of cooperating nodes. Finally, we believe that the performance measurements presented show the extent to which the VAXcluster system has succeeded in implementing efficient communications architecture. These numbers are particularly impressive when considering that VMS is a large, general-purpose operating system.

Kreutzer and Hakimi [1988] have studied the distributed diagnosis system with respect to user's observation that needs to know the status of the system in order to repair the faulty units and/or obtain the results of computing tasks from the fault free units.

Laprie et. al.[1990] have found some Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures. They have mentioned that Systems in which one piece of hardware supports multiple software are subject to Software failures and require architectures that tolerate both software faults. They have found that there are two main approaches to detecting errors caused by design faults:

(i) Acceptance tests of the results via executable assertions. These assertions are generalized, formalized versions of likelihood checks used in process control.

(ii) Diversified design, so that the results of two software variants can be compared. The best-documented techniques for tolerating software design faults are the recovery block **(RB)** approach5 and N-version programming (**NVP**). They have designed two architectures for tolerating a single fault and for two consecutive faults**.** They have analyzed software-fault-tolerant architectures and their analysis emphasizes the distinctions among the different sources of failures - independent and related faults in the variants and the decider - and assumes that only one type of fault can cause errors during each execution. They have also evaluated hardware and software architectures. And they have assumed the behavior of the architectures in their modeling that only one type of fault can cause an error (either hardware or software) during each execution; the variant is not discarded after error detection and recovery, but is given the new input data at the next step (that is, software faults are soft); and the

hardware components and the software- fault-tolerant architectures have constant failure rates.

Ramanathan et. al.[1990] have studied Fault Tolerant Clock Synchronization in Distribution System. They have concentrated mainly because many applications require consistent view of time across all nodes of a distributed system and this can be achieved by clock synchronization only. They have proposed a solution on clock synchronization take either software or hardware approach. The software approach is economical and flexible but additional message must be exchanged solely for synchronization. The hardware approach on the other hand uses special hardware at each node to achieve a tight synchronization with minimal time overhead. Their work compares and contrasts existing fault-tolerant clock synchronization algorithms. The worst case clock skews guaranteed by representative algorithms are compared, along with other important aspects such as time, message and cost overhead imposed by the algorithms special emphasis is given to more recent developments such as hardware-assisted software synchronization, probabilistic clock synchronization and algorithms for synchronizing large partially connected distributed systems. In the study they found the basic idea of software synchronization algorithm is that each node has a logical clock that provides a time base for all activities on that node. This logical clock is derived from the hardware clock on that node, through it usually has a much larger granularity than the hardware clock. On the other hand the principle of hardware synchronization algorithm is that of a phase-locked loop. The hardware clock at each node is an output of the voltage-controlled oscillator. The voltage applied to the oscillator comes from a phase detector whose output is proportional to the phase error between the phase of its clock and a reference signal generated by using the other clocks in the system. Besides these two synchronizations to remove the inherent limitations of both the hardware and software approaches they have proposed a hybrid synchronization scheme that strikes a balance between the clock skews attainable and the hardware requirement. Their scheme is particularly suitable for large, partially connected homogeneous distributed systems with point-to-point interconnection topologies such as hyper cubes or meshes. The hybrid scheme is cost-effective and achieves a reasonably tight synchronization. It is also suitable for synchronizing large, partially connected system and hence most viable scheme for future distributed systems.

Elmootazbellah et. al.[1991] have worked to measure performance of consistent checkpointing. As consistent checkpointing protocols can be used to provide transparent fault tolerance for long-running distributed application programs. They had given a preliminary report on the performance measurement and implementation of consistent checkpointing on a distributed system running on 16 diskless workstations. These measurement included the effect of consistent checkpointing on the running time of six computed- bound distributed application programs. They found consistent checkpointing performs very well for five of the applications in their study, for the sixth program, which rapidly modifies large amounts of memory during its execution, the overhead of consistent checkpoint is higher. The primary factor affecting the performance of consistent checkpointing is the performance of stable storage on which the checkpoints are recorded. They found that the number of messages required by the consistent checkpointing protocol is not a significant factor in the overall performance in their environment.

Beven Keith and Binley Andrew[1992] described a methodology for calibration and uncertainty estimation of distributed models based on generalization likelihood measures. In their work the Generalized Likelihood Uncertainty Estimation (GLUE) procedure provided a formal framework for taking into account some of the particular difficulties associated with the calibration and application of distributed hydrological models.

Cristian Flaviu[1993] has proposed a small number of basic concepts that can be used to explain the architecture of fault-tolerant distributed systems and discussed a list of architectural issues that will find useful to consider when designing or examining such systems. For each issue he has presented know solutions and design alternatives. He discussed their relative merits and gave examples of systems which adopt one approach or the other. His aim was to introduce some order in the complex discipline of designing and understanding fault-tolerant distributed systems.

Bianchini Jr. [1994] has studied pertains to an adaptive distributed system for fault tolerance. The system is comprised of a network. The system is also comprised of N nodes, where N is greater than or equal to 3 and is an integer, and a node is able to execute an algorithm in communication with the network.

Chelizatn Lining and Avizienis Algridas [1996] used a methodology for implementing N-Version programming which is relatively simple and can be generalized to other similar applications.

Gartner Felix C.[1999] has used a formal approach to structure the area of fault tolerant

distributed system. The advantage of formal approach, however also lie in the fact that it reveals the inherent limitations of fault tolerance methodology and their interaction with system models.

Ghosh Sudipta and Mathur Aditya P.[1999] have described the issues in testing component based distributed systems related to the heteroginity of performance, differences in languages, concurrency,scalability and underlying communication protocols. They have proposed an interface based testing methodology for testing components and systems.

Lakshmanan et. al.[2000] have proposed a model which has wide applicability since faults and hence their propagation can be interpreted in a number of ways depending on the system consideration.

Bagchi et. al.[2001] had studied Dependency Analysis in Distributed Systems using Fault Injection and application to Problem Determination in an e-commerce Environment. They have proposed using fault injection as the perturbation tool for dynamic dependency discovery and problem determination. They have described a method for characterizing dependencies of transactions on the system resources in a typical e-commerce environment, and show how it can aid in problem diagnosis. The method is applied to an application server middleware platform, running end-user activity. The method was applied to a real-world web-based e-commerce environment to illustrate how it can aid in root cause determination for an end-user visible problem. One problem with the approach is that it is invasive in nature and hence needs careful consideration of the conditions under which it can be applied. The system is already in a malfunctioning mode. However, it is possible that the fault injection campaign is extremely invasive and completely halts the operation of the system.

Chessa S. and Shanti P.[2001] had addressed the problem of fault identification in ad-hoc networks. They had presented a new comparison-based diagnostic model based on the one-to-many communication paradigm which takes advantage of the shared nature of communication typical of multi-hop packet radio networks. Two implementations of the model were presented. The first implementation assumes that the network topology is fixed and does not change during the diagnosis. Under this scenario, hard faults can be detected using a timeout, and efficient diagnosis protocols can be easily designed. If the fixed topology assumption is released, thus taking into account a relevant feature of ad-hoc networks, the "diagnostic efficiency" of the model decreases notably: hard-faults cannot be detected and fault-free nodes are no longer guaranteed to correctly diagnose all their neighbors within a certain time. In the second implementation they allowed the system topology to change during diagnosis. They show that the ability of diagnosing faults under the scenario decreases, meaning that mobility significantly reduces the quality of the diagnosis returned by a diagnosis protocol.

Caruso et. al.[2002] had proposed a novel approach aimed at evaluating the diagnosability of regular systems under the PMC model is introduced. The diagnosability is defined as the ability to provide a correct diagnosis, although possibly incomplete. This concept is somehow intermediate between one-step diagnosability and sequential diagnosability.

Chessa S.and Santi P.[2002] had considered the problem of identifying faulty (crashed) nodes in a wireless sensor network. They had studied the problem is of fundamental importance in those applicative scenarios of wireless sensor networks in which battery replacement is feasible. The diagnostic information gathered by operational sensors can be used by an external operator for the sake of network reconfiguration and/or repair, thus extending network lifetime. A fault diagnosis protocol specifically designed for wireless sensor networks is introduced and analyzed. The protocol is proved to be optimal and energy efficient under certain assumptions. The system used by them is composed by *n* sensors, also called *nodes*, which communicate via radio transceivers. Nodes are homogeneous and equipped with a limited energy supply. Any node in the system is a potential *sink*, i.e. it can be used by an external operator to access the information gathered by the network. Sensors can be in one of two states: *faulty* or *fault-free*. Faulty nodes are unable to communicate with the rest of the system, either due to a crash or to battery depletion. This means that faults are *permanent*, i.e. nodes remain faulty until they are repaired and/or replaced. We provided bounds on the maximum number of faults that can be tolerated and on the minimum number of bits to be exchanged by any correct diagnosis protocol. They also introduced the WSNDiag diagnosis protocol specifically designed for wireless sensor networks, and we proved that WSNDiag provides optimal diagnosis, that is, it is has maximal diagnosability while minimizing the number of bits exchanged (hence, under certain assumptions, the energy consumption) for the purpose of diagnosis.

Su et. al.[2002] had studied an traditional centralized network management system and found that traditional centralized network management solutions do not scale to present-day large-scale

computer/communication networks. They suffer from certain other drawbacks to a point of failure and hence lack of fault tolerance, and heavy communication costs associated with the central manager. It has been recognized that decentralization/distributed solutions can solve some of the problem associated with centralized solutions for this reason; there has been considerable interest in the recent past on distributed/decentralized network management applications. Their work in the paper has been motivated by this research trend in network management. They presented the design and evaluation of SNMP-based distributed network fault detection/monitoring system. The design involver the integration of their recently developed ML-ADSD algorithm for diagnosis of faults In a distributed system of processors into the SNMP framework The ML-ADSD algorithm uses the multilevel paradigm and is scalable in the sense that only minor modifications will be required to adapt the algorithm to network of varying sizes. The system allows processors to fail and/or recover during the process diagnosis. Thus the system has fault tolerance capability. They had demonstrated the application of the system by implementing it on an Ethernet network of 32 machines. Our results established that the diagnosis latency (or time to termination) is much better than the latency of earlier solutions. Also, the bandwidth utilization of our system is very insignificant, thereby demonstrating the practicability of deployment of the system in a real network environment. Thus in this work they had successfully integrated in three modern disciplines: network management, distributed computing and system level diagnosis.

Venkatasubramanian et. al.[2002] in their final part, they had discussed fault diagnosis methods that are based on historic process knowledge. They also compared and evaluated the various methodologies reviewed in this series in terms of the set of desirable characteristics they proposed in their earlier study. This comparative study reveals the relative strengths and weaknesses of the different approaches. One realizes that no single method has all the desirable features one would like a diagnostic system to possess. It is their view that some of these methods can complement one another resulting in better diagnostic systems. Integrating these complementary features is one way to develop hybrid systems that could overcome the limitations of individual solution strategies. The important role of fault diagnosis in the broader context of process operations is also outlined. They also discussed the technical challenges in research and development that need to be addressed for the successful design and implementation of practical intelligent supervisory control systems for the process industries.

Legrand et. al.[2003] had recognized two important limitations of simulations used for scheduling research:

(i) There is no simulation standard;
(ii) Simulation methodologies are not appropriate for modern distributed computing platforms such as the Grid.

They had presented SimGrid v2 which enables scalable, configurable, extensible, and fast simulations for investigating novel scheduling techniques for these platforms. SimGrid has already been used successfully and the SimGrid user community is currently undergoing a dramatic expansion. SimGrid is also used for educational purposes in a course on Parallel Algorithms and Architectures. Future directions include mechanisms to import Brite topologies and associate them link characteristics in an ENV fashion for SimGrid. They had also created a repository where SimGrid users can download pre-generated topologies. A difficult question is that of simulation validation and few projects have addressed it successfully. The rationale is that a simulation implements a model that makes it possible to explore and reason about systems. Whether the model actually reflects reality is somewhat of a different question. SimGrid implements a number of models that have been widely accepted in the scheduling research community. Nevertheless, they had also designed and implemented models that they believed are more realistic and for which they had conducted initial validation experiments.

Han et. al.[2003] they found that with efficient scheduling algorithms and software fault tolerant deadline mechanisms, one can design a system that meets its task timing constraints while tolerating software faults. They considered the problem of scheduling real-time periodic tasks using the deadline mechanism to provide software fault tolerance. They had proposed a scheduling algorithm based on the last chance philosophy to schedule the alternates as late as possible and leave as much room as possible for executing the primaries before their deadlines. Their basic algorithm makes significant performance improvements by integrating two very simple but elegant heuristics. The simulation results have shown that, in some worst cases, specifically, the cases with high failure probability and/or high task utilization, the modified algorithm can make an impressive performance enhancement; for example, the percentage of successful primaries is increased by 300 percent in certain cases. In the final part of this

study, they also proposed a more adaptive algorithm, enabling their algorithm to deal with dynamic task systems.

Fujimoto[2003] in his area of distributed simulation systems had studied an overview of technologies concerned with distributing the execution of simulation programs across multiple processors. He found, particular emphasis is placed on discrete event simulations. The study here is focused on time management, a central issue concerning the synchronization of computations on different processors. Time management algorithms broadly fall into two categories, termed conservative and optimistic synchronization. A survey of both conservative and optimistic algorithms is presented focusing on fundamental principles and mechanisms. Finally, time management in the High Level Architecture (HLA) is discussed by the author as a means to illustrate how this standard supports both approaches to synchronization.

Renesse et. al.[2003] had studied in the area of Scalable management and self-organizational. As they are emerging as central requirements for a generation of large-scale, highly dynamic, distributed applications. They had developed an entirely new distributed information management system called Astrolabe. Astrolabe collects large-scale system state, permitting rapid updates and providing on-the-fly attribute aggregation. This latter capability permits an application to locate a resource, and also offers a scalable way to track system state as it evolves over time. The combination of features makes it possible to solve a wide variety of management and self-configuration problems. This paper describes the design of the system with a focus upon its scalability. After describing the Astrolabe service, we present examples of the use of Astrolabe for locating resources, publish-subscribe, and distributed synchronization in large systems. Astrolabe is implemented using a peer-to-peer protocol, and uses a restricted form of mobile code based on the SQL query language for aggregation. This protocol gives rise to a novel consistency model. Astrolabe addresses several security considerations using a built-in PKI. The scalability of the system is evaluated using both simulation and experiments; these confirm that Astrolabe could scale to thousands and perhaps millions of nodes, with information propagation delays in the tens of seconds.

Subbiah and Blough[2004] had studied the problem of distributed diagnosis in the presence of dynamic failures and repairs are considered. To address this problem, they used the notion of bounded correctness is defined. Bounded correctness is made up of three properties: bounded diagnostic latency, which ensures that information about state changes of nodes in the system reaches working nodes with a bounded delay, bounded start-up time, which guarantees that working nodes determine valid states for every other node in the system within bounded time after their recovery, and accuracy, which ensures that no spurious events are recorded by working nodes. It is shown that, in order to achieve bounded correctness, the rate at which nodes fail and are repaired must be limited. This requirement is quantified by defining a minimum state holding time in the system. Algorithm Heartbeat Complete is presented and it is proven that this algorithm achieves bounded correctness in fully-connected systems while simultaneously minimizing diagnostic latency, start-up time, and state holding time. A diagnosis algorithm for arbitrary topologies, known as Algorithm Forward Heartbeat, is also presented. Forward Heartbeat is shown to produce significantly shorter latency and state holding time than prior algorithms, which focused primarily on minimizing the number of tests at the expense of latency.

Zhang et. al.[2004] had studied An Information Based Diagnostic Approach for Adaptive Fault-Tolerant Control of Nonlinear Uncertain Systems. In this paper, they had presented a unified methodology for fault diagnosis and accommodation in a class of nonlinear systems. The proposed FTC architecture consists of an on-line monitoring module used to detect and to isolate faults, and a controller module to accommodate the effects of faults on the basis of the fault information obtained by the fault-diagnosis procedure. An adaptive tracking design has been developed that uses neural networks to approximate the unknown fault function. The fault-tolerance has been enhanced by use of adaptive bounding design techniques. Under certain assumptions, the stability of the proposed robust FTC scheme has been rigorously established by using the Lyapunov synthesis approach. The extension of the proposed integrated approach to fault diagnosis and controller reconfiguration to a larger class of nonlinear systems and faults deserves further research. Moreover, considerable effort is devoted to generalizing the methodology to the case where not all state variables are available for measurements; this is clearly very important from an application point of view.

Xu Y. and Qi H. [2004] had reported the development of an energy-efficient, high-performance distributed computing paradigm to carry out Collaborative Signal and Information Processing (CSIP) in sensor networks using mobile agents in the study. They had focused on the development of three

distributed computing paradigms for Collaborative Signal and Information Processing in sensor networks. Performance of the client/server-based paradigm and the mobileagent-based paradigm through both mathematical modeling and simulation was first compared in the study. The processing code was moved to the sensor nodes through mobile agents, in contrast to the client/server-based computing, where local data are transferred to a processing center. In the paper they had mentioned that though the client/server paradigm had been widely used in distributed computing, the many advantages of the mobile agent paradigm made it more suitable for sensor networks. Their paper first presented simulation models for both the client/server paradigm and the mobile agent paradigm. They had designed three metrics for performance evaluation: execution time, energy usage, and energy*delay. They found that the mobile agent paradigm performs much better when the number of nodes is large while the client/server paradigm is advantageous when the number of nodes is small. They had proposed a cluster-based hybrid computing paradigm to combine the advantages of these two paradigms and concluded that the cluster-based hybrid computing provides an energy-efficient and high-performance solution to CSIP.

Balazinska et. al.[2005] had studied Fault Tolerance in the Borealis Distributed Stream Processing System. They had presented a replication-based approach to fault tolerant stream processing that handles node failures, network failures, and network partitions. Their approach uses a new data model that distinguishes between stable tuples and tentative tuples, which result from processing partial inputs and may later be corrected. Our approach favors availability but guarantees eventual consistency. Additionally, while ensuring that each node processes new tuples within a predefined delay, X, their approach reduces the number of tentative tuples, when possible. To ensure consistency at runtime, they introduce a data-serializing operator called SUnion. To regain consistency after failures heal, nodes reconcile their states using either checkpoint/redo or undo/redo. They implemented the approach in Borealis and showed several experimental results. For short failures, SPE nodes can avoid inconsistency by blocking and looking for a stable upstream neighbor. For long failures, nodes need to process new inputs both during failure and stabilization to ensure the required availability. Checkpoint/redo leads to a faster reconciliation at a lower cost compared with undo/redo. Many stream processing applications prefer approximate results to long delays but eventually need to see the correct output streams. It is important that failure-handling schemes meet this requirement. We view this work as an important first step in this direction.

Treaster M.[2005] had done a survey which presents an overview of fault-tolerance techniques in large scale cluster computing systems. These techniques can be grouped into two categories: protection for the cluster management hardware and software infrastructure, and protection for the computation nodes and the long-running applications that execute on them. Cluster management hardware and software fault-tolerance typically makes use of redundancy, due to the relatively small number of components that need to be duplicated for this approach. When a component fails, the redundant components take over the responsibilities of the failed parts. Redundancy can also be used for fault detection by comparing the outputs produced by each replica and looking for discrepancies. Cluster applications are protected from faults using check pointing and rollback recovery techniques. Each process cooperating in the application periodically records its state to a checkpoint files in reliable, stable storage. In the event of a process failure, the application state is restored from the most recent set of checkpoints. There are a variety of protocols that have been developed to determine when processes should record checkpoints and how to restore the application state. Fault tolerance solutions can be implemented in a variety of forms. This includes software libraries, special programming languages, compiler or preprocessor modifications, operating system extensions, and system middleware. Each method has its own tradeoffs in terms of power, portability, and ease of use.

Saikia L. P.[2006] had studied arbitrary networks, distributed algorithm for fault diagnosis uses parallel dissemination of fault event information to minimize the information latency in the network. Algorithm made evident the tradeoff between information latency and message overhead. In networks consisting at least partially of point-to-point communication links, it is argued that the increased message overhead that leads to optimal information latency is tolerable.

Perumalla K. S.[2006] had studied parallel and distributed simulation on the basis of traditional techniques and recent advances the presented an overview of parallel and distributed simulation systems, their traditional synchronization approaches and a case study using the HLA standard interface and implementation. Recent advances, such as scalability to supercomputing platforms and novel rollback techniques have been presented. The interaction of parallel simulation with newly

emerging hardware architectures is outlined. The future outlook seems to warrant focus on needs from lager scale simulation scenarios to be achieved on high-end computing plat-forms. There is also interest on high-performance simulations on low-end platforms such as using the multi-core architectures, GPGPUs and other co-processor-based systems. However, practical challenges remain to be explored, including: (a) wide spectrum of network latencies (b) highly dynamic participation by processors (c) semantics and implementations of always-on presence for large simulation.

Fourlas G. K.[2007] had studied an Approach towards Fault Tolerant HybridControl Systems. He stated that the notion of fault tolerant hybrid control systems is an important problem to deal with, since faults appearance is inevitable in all real systems and especially in hybrid systems. The most significant challenge arises from the complexity of the system, which forces designers to develop more sophisticated schemes. In this paper we have introduced the underlying concepts for our approach to fault tolerant hybrid control systems using the fault diagnosis of hybrid systems technique [41], to prevent local faults to be developed into system failures. The method can also be used to evaluate the degree of system performance degradation after occurrence of a fault. This approach was illustrated via a simple application to an electric power transmission system where the interaction between the continuous dynamics and an appropriate corrective action (in this case a 50% load shedding in order to avoid a blackout - discrete event dynamic) has been presented. Because of the limited space reasons, only a brief description of the theoretical fault diagnosis approach is given in this paper. Current research includes the fault propagation the handling of multiples faults as well as of successive faults. An important issue for investigation is the algorithmic complexity, in cases where large number of components and subsystems are present.

Hursey et. al.[2007] had studied the Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI. They found that, to be able to fully exploit ever larger computing platforms, modern HPC applications and system software must be able to tolerate inevitable faults. Historically, MPI implementations that incorporated fault tolerance capabilities have been limited by lack of modularity, scalability and usability. They had presented the design and implementation of an infrastructure to support checkpoint/restart fault tolerance in the Open MPI project. They had identified the general capabilities required for distributed checkpoint/restart and realized these capabilities as extensible frameworks within Open MPI's modular component architecture. Our design features an abstract interface for providing and accessing fault tolerance services without sacrificing performance, robustness, or flexibility. Although our implementation includes support for some initial checkpoint/restart mechanisms, the framework is meant to be extensible and to encourage experimentation of alternative techniques within a production quality MPI implementation.

Saikia L. P. and Hemachandran K.[2007] had studied the problem of distributed diagnosis in arbitrary network failures and repairs in their work. A number of investigations had been attempted to extend traditional notions of "fault-tolerant computing", to deal with the problem of failures, which affect the facilities of distributed systems and computer networks by author. According to the authors, as distinguish of diagnostic responsibility requires the flow of diagnostic information through the network, and the faulty facilities themselves may participate in this flow and may alter, destroy, or generate erroneous diagnostic information in the process, the whole diagnostic procedure itself becomes quite complex. Their purpose of this study is to simulate a distributed system and carry out fault diagnosis under Arbitrary Network topologies. In their study since "system level diagnosis" is one of the steps in the process of building "distributed fault-tolerant systems", reliability of such a system depends heavily on proper functioning of the diagnosis algorithm. They used JAVA with Console.Java program to create multiple windows, for the use of distributed system simulation each one representing a different system node. In their simulation model distributed diagnosis algorithms had been simulated. As per implementation of the algorithm for arbitrary networks concerned, 15 nodes have been considered as 15 consoles in the study. Very interestingly they had seen visually each of the nodes running on a respective Console. In each of the Console, multiple threads are running by which execution of the respective algorithm's output was viewed. As there was scrolling facility in each of the Console, they could view each of the steps running in a particular node. In their study they found that distributed algorithm for fault diagnosis that uses parallel dissemination of fault event information to minimize the information latency in the network. Even though it works for arbitrary networks, it assumes no link failures. The outcome of the study was that a newly repaired node can rejoin the system without relying on other nodes to first detect that it has been repaired; equivalently, faulty nodes do not have to be periodically tested.

Zhou W.[2010] had studied the area of Fault Management in Distributed Systems. In his study he had presented a survey of these mechanisms and systems, and taxonomize them according to the techniques adopted and their application domains. Based on four representative systems (Pip, Friday, PeerReview and TrInc), he discussed various aspects of fault management, including fault detection, fault diagnosis and evidence generation. Their strength, limitation and application domains are evaluated and compared in detail.

Zaharia et. al.[2012] had presented Resilient Distributed Datasets (RDDs), a distributed memory abstraction which provides programmers to perform in-memory computations on large clusters in a fault-tolerant manner. They had found that RDDs can express a wide range of parallel applications, moreover many specialized programming models that have been proposed for iterative computation, and new applications that these models do not capture can also be expressed. According to the authors RDDs offer an API based on coarse-grained transformations that lets them recover data efficiently using lineage unlike existing storage abstractions for clusters, which require data replication for fault tolerance. In the paper they had mentioned that RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. They had showed that including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these models do not capture, RDDs are expressive enough to capture a wide class of computations. Using and evaluated through a variety of user applications and benchmarks, the researchers implemented RDDs in a system called Spark.

Shamsudeen. E and Sundaram V.[2013] had studied the issues with orphan computation in distributed Computing System. They have studied how orphan processes are born and their effects on the system. In their study they have mentioned the problem raised by the orphan processes like wastage of computer resources, data inconsistency etc. in their paper Issues with Orphan Computations in Distributed Computing Systems one specific problem situation of ATM transaction was mentioned and how orphan process can create hazards in common man's day to day life has been discussed.

Mukhin V. and Volokyata A.[2013] had studied the safety mechanism of Distributed Computing System. They had mentioned the subdivision of safety problems in DCS are as Integration Problems, Interaction Problems and Problems of trusted relations. They had also proposed a model of safety in DCS, which support the complex protection of various DCS segments and resource forming the virtual computing environment. The model comprises of protection of data transmission protocol, data transmission, safe communication channels, authentication and transfer/display of safety certificates, access rights to resources delimitation, privacy support of the users, trust in DCS, security control of DCS, security tools of DCS and modern standards in the field of DCS safety as components. Moreover they had mentioned that to reduce the vulnerability level in DCs and to avoid the losses of critical information, additional security mechanisms like security risk analysis in DCS should be taken.

5. Conclusion

Hardware, software and networks cannot be totally free from failures. Fault tolerance is a non-functional requirement that requires a system to continue to operate, even in the presence of faults. Distributed systems can be more fault tolerant than centralized systems. Agrement in faulty systems and reliable group communication are important problems in distributed systems. Replication of Data is a major fault tolerance method in distributed systems. Recovery is another property to consider in faulty distributed environments.

References:

1. Ramamoorthy C.V., "A structural theory of machine diagnosis", Spring Joint Computer Conference Proc. Montvale, NJ, pp. 743-756, 1967
2. Short R.A. "The Attainment of reliable digital systems through the use of redundancy-A Survey" IEEE Computer Group News, pp 2-17, Mar, 1968
3. Avizienis A., "Fault tolerance computing-An overview" IEEE Computer, vol.4, pp 5-8 Jan/Feb 1971
4. Khul J.G. and Reddy S.M., "Fault-diagnosis in fully distributed system" in Proc. 11th Int. Symp. Fault Tolerant Computing, pp 100-105, June1981.
5. Bagchi A. and Hakimi S.L. "An optimal algorithm for distributed system level diagnosis" in Proc. 21th Int. Symp. Fault Tolerant Computing, pp 214-221, June 1991
6. Hayes John P., "A Graph Model for fault tolerant Computing System" IEEE Transactions on Computers, Vol. C-25, No. 9, pp. 875-884, September,1976
7. Lamport Leslie, "Using time instead of timeout for fault tolerant distributed system"

ACM Transactions on Programming Languages and Systems, Vol. 6, No. 2 ,pp 254-280,April 1984

8. Holt C.S. and Smith J. E., "Self Diagnosis in distributed systems" IEEE Transactions on Computers, Vol. 34 No. 1, pp, 19-32, January1985
9. Nancy P. Kronenberg, Henry M. Levy and William D. Strecker," VAXclusters: A Closely-Coupled Distributed System" ACM Transactions on Computer Systems, Vol. 4, No. 2, pp 130-146, May 1986.
10. Kretuzer Steven.E. and Hakimi S.Louis., "Distributed Diagnosis and system user", IEEE Transaction on Computers, Vol. 37, No. 1, pp. 71-78, January 1988.
11. Laprie Jean-Claude, Arlat Jean, Bounes Christian and Kanoun Karama, "Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures", IEEE Computers, pp 39-51, July 1990.
12. Ramanathan Parameswar, Shin Kang G. and Butler Ricky W., "Fault-Tolerant Clock Syncronization in Distributed System", IEEE Computers, pp 33-42, October 1990.
13. Elmootazbellah N. Elnozahy, David B. Johnson and Willy Zwaenepoel, "Measured Performance of Consistent Checkpoints" December 1991.
14. Beven Keith and Binley Andrew, "The Future of Distributed Models: Model Calibration And Uncertainty Prediction", Hydrological Processes, Vol. 6, pp 279-298, 1992.
15. Cristian Flaviu, "Understanding Fault-Tolerant Distributed System", May 1993.
16. Bianchini Jr., "Adaptive distributed system and method for fault tolerance" 1994.
17. Chelizatn Lining and Avizienis Algridas, "N-version programming: A Fault-Tolerance approach to reliability of software operation", IEEE proceeding of FTCS-25, Vol. 3, 1996.
18. Gartner Felix C., "Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments", ACM Computing Surveys, Vol. 31, No. 1, March 1999.
19. Ghosh Sudipta, Mathur Aditya P., "Issues in Testing Distributed Component Based Systems", March 1999.
20. Lakshmanan K.B., Rosenkrantz D, J. and . Ravi S. S., "Alarm Placement in Systems with Fault Propagation", Pacific Rim Int. Sym. On Fault Tolerant Systems, December 1995.
21. Bagchi S., Kar G. and Hellerstein J., "Dependency Analysis in Distributed Systems using Fault Injection: Application to Problem Determination in an e-commerce Environment", 12th International Workshop on Distributed Systems: Operations and Management, October 2001.
22. Chessa S. and Shanti P., "Comparison-Based System-Level Fault Diagnosis in Ad-Hoc Networks", 2001.
23. Caruso A., Chessa S., Maestrini P. and Santi P., "Diagnosability of regular systems", Elsevier Science, 2002.
24. Chessa S. and Santi P., "Crash Faults Identification in Wireless Sensor Networks", 2002.
25. Su M. Shan, Thulasiraman K. and Das A., "A Scalable On-Line Multilevel Distributed Network Fault Detectioflonitoring System Based on the SNMP Protocol", IEEE Computers, pp 1660-1664, 2002.
26. Venkatasubramanian V. , Rengaswamy R., Kavuri S. N. and Yin K., "A review of process fault detection and diagnosis Part III: Process history based methods", Elsevier Science, 2002.
27. Legrand A., Marchal L. and Casanova H., "Scheduling Distributed Applications:the SimGrid Simulation Framework", Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid 2003.
28. Han C.C., Shin K. G. and Wu J., "A Fault-Tolerant Scheduling Algorithm for Real-Time Periodic Tasks with Possible Software Faults", IEEE Computers 2003.
29. Fujimoto R. M., "Distributed Simulation Systems", Proceedings of the Winter Simulation Conference, 2003.
30. Renesse R. V., Birman K. P., and W. Vogels, "Astrolabe: A Robust and ScalableTechnology for Distributed System Monitoring, Management, and Data Mining", ACM Transactions on Computer Systems, Vol. 21, No. 2, Pages 164–206, May 2003.
31. Subbiah A. and Blough D. M., "Distributed Diagnosis in Dynamic Fault Environments", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 15, NO. 5, pp 453-467 MAY 2004.
32. Zhang X., Parisini T. and Polycarpou M. M., "Adaptive Fault-Tolerant Control of Nonlinear Uncertain Systems: An

Information-Based Diagnostic Approach", IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 49, NO. 8, pp 1259-1274 AUGUST 2004.

33. Xu Y. and Qi H., " Distributed Computing Paradigms for Collaborative Signal and Information Processing in Sensor Networks ", Elsevier Science,2004.
34. Treaster M., "A Survey of Fault-Tolerance and Fault-Recovery Techniques in Parallel Systems", 2005.
35. Balazinska M., Balakrishnan, Madden H., S. and Stonebraker M., "FaultTolerance in the Borealis Distributed Stream Processing System, ACM, June 2005.
36. Saikia L. P., "Distributed system level fault diagnosis", NWTAC, 2006.
37. Perumalla K. S., "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances", Proceedings of the Winter Simulation Conference, 2006.
38. Fourlas G. K., "An Approach Towards Fault Tolerant Hybrid Control Systems", Proceedings of the 15th Maditerranean Conference, July 2007.
39. Hursey J., Squyres J. M., Mattox T. I., Lumsdaine A., "The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI", IEEE computers, 2007.
40. Saikia L. P. and Hemachandran K., "Simulation of System Level Diagnosis in Distributed Arbitrary Network", JATIT, 2007.
41. Zhou W., "Fault Management in Distributed Systems", 2010.
42. Fourlas G.K., Kyriakopoulos K.J. and Krikelis N.J., "A Theoretical Diagnosis Approach Applied to a Power Transmission System", Proceedings of the 43rd IEEE Conference on Decision Control, Paradise Island, Bahamas, December 2004.
43. Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauley M., Franklin M. J., Shenker S. and Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for
In-Memory Cluster Computing", 2012.
44. Shamsudeen. E and Sundaram V., "Issues with Orphan Computations in Distributed Computing Systems" International Journal of Computer Applications(0975-8887) Volume 62, No. 20, January 2013.
45. Mukhin V. and Volokyata A., "Integrated Safety Mechanisms Based on Security Risks Minimization for the Distributed Computer Systems", I.J. Computer Network and Information Security, 2, Pages 21-28, 2013.