Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 6, June 2015, pg.829 – 836

SURVEY ARTICLE

Fault Tolerance in Distributed Real Time Environment: A Survey

Neelutpol Gogoi[1], Lakshmi Prasad Saikia[2]
[1]PhD Research Scholar, Dept. of Computer Sc.& Engg., Assam Down Town University, India
[2]Professor, Dept. of Computer Sc. & Engg., Assam Down Town University, India
[1] gogoi.neelutpol@gmail.com; [2] lp_saikia@yahoo.co.in

Abstract— The basic requirements of a real time environment are timeliness and reliability. In distributed Real-time environment, attaining timeliness and reliability even in the condition of faults is a real challenge. Fault tolerance is the ability of a system to endure its functionality by tolerating any faults or errors that occur. In distributed real time environment a fault may occur due to numerous reasons like communication failure, resource or hardware failure or process faults which occur due to shortage of storage of resource, software bugs etc. But, even in the condition of faults, in distributed real time environment, the task should be completed within the allotted time constrains. This paper aims at providing a better insight in faults, fault tolerance and the techniques for attaining fault tolerance in distributed real time environment.

Keywords— Distributed system, real time system, fault, fault tolerance, redundancy, Load balancing, Replication.

I. INTRODUCTION

Distributed system is a computing concept that refers to a collection of independent computers which appears to its users as a single coherent system. In its most general sense, it refers to multiple autonomous computer systems interacting through a communication channel and working together on a single problem. In distributed computing, a single problem is divided into many parts, and each part is solved by different independent but connected computers [13]. As long as the computers are networked, they can communicate with each other to solve the problem, performing like a single entity.

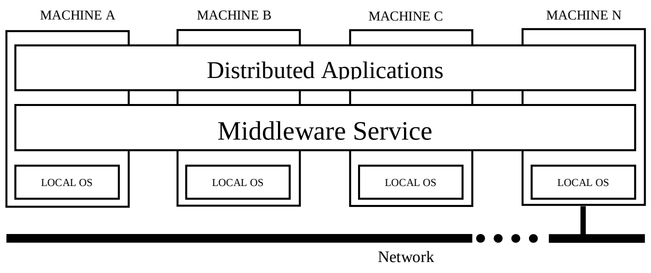

Fig. 1. A distributed system

As shown in the fig.1 a distributed computing architecture consists of a number of client machines connected to a common network. Each machine possesses its own local operating system, applications and resources other than the common distributed system layer which is also called the middleware. The characteristics of distributed systems are resource sharing, openness, scalability, transparency and most importantly fault tolerance, with which it achieves its basic objectives to maximize performance by connecting users and IT resources in a cost-effective, transparent and reliable manner.

A real-time system is one whose correctness is based on both the correctness of the outputs and their timeliness. A real-time system must process information and produce a response within the specified time constrains, that is, the system is time critical, else risk severe consequences like degradation of results or catastrophic system failure. Which means, in a system with real-time constraints, it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or else useless.

If the real-time computer system is distributed, it consists of a set of nodes (computers) interconnected by a real-time communication network. In a distributed system several nodes work together towards a common goal and in real-time systems we need to find a result within a deadline [25]. With a distributed real-time system (such as nuclear systems, robotics, air traffic control systems, grid etc.) we combine these and have a system where several nodes work together with a deadline towards a common goal. A fault in distributed real-time system (for example an airplane control system that is a combination of sensors, actuators and different automated systems all controlled from the cockpit) can lead to catastrophic results if not properly detected and recovered at time. These systems must function with high availability even under hardware and software faults. Fault-tolerance is the important technique used to maintain dependability in these systems. Numerous methods are being adopted to acquire fault tolerance in distributed real-time environments. Hardware and software redundancy are two well-known effective methods. This paper will give an insight on faults and fault tolerant systems in distributed real time environment.

II. FAULT AND FAULT TOLERANCE

Fault is a defect or flaw in any system components which leads to an error and finally a failure of the system [14]. There are different types of faults which can occur in a real time distributed system. These faults can be broadly classified as: Network faults, Physical fault, media faults, process faults, Service expiry faults. Network faults occur in a network due to network partition, packet loss, communication failure etc. Physical faults can occur in hardware like fault in CPUs, memory fault etc. Media faults occur due to media head crashes. Process faults occur due to shortage of resources, software bugs etc. Service expiry fault occur in a network because the service time of a resource may expire while application is using it [4] [15] [46].

By their origin, a fault may be physical, human, internal, external, conception, operational. Faults can also be classified with respect to time or persistence as follows:
Permanent: These failures occur by accidently cutting a wire, power breakdown and so on which can cause major disruptions and some part of system may not be functioning as desired. *Intermittent*: These failures appear occasionally. *Transient*: They are caused by some inherent fault in the system. However, these failures are corrected by retrying roll back the system to previous state such as restarting software or resending a message. Not all faults cause immediate failure: faults may be *latent* (activated but not apparent at the service level), and later become *effective*. Fault-tolerant systems attempt to detect and correct latent errors before they become effective. In real time system, main focus is on hardware fault tolerance. Not all faults cause immediate failure:

faults may be *latent* (activated but not apparent at the service level), and later become *effective*. Fault-tolerant systems attempt to detect and correct latent errors before they become effective.

Fault-tolerance is the ability of a system to maintain its functionality, even in the presence of faults. The ability of a system to recover automatically from unexpected faults is called Self-stabilization [16]. A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails [1]. The fault tolerance can be generally obtained by the following two techniques.

- *error processing* (to remove errors from the system's state), which can be carried out either with *recovery* (rolling back to a previous correct state) or with *compensation* (masking errors using the internal redundancy of the system);
- *fault treatment* (to prevent faults from being activated again), which is carried out in two steps: *diagnostic* (determining the cause, location, and nature of the error) and then *passivation* (preventing the fault from being activated again).

The distributed real-time system architecture provides a significant potential for fault-tolerance.

III. RELATED WORKS

The idea of distributing resources within a computer network was first started with the use of data entry terminals on mainframe computers, then moved into minicomputers and is now possible in personal computers and client-server architecture with more tiers. The use of concurrent processes that communicate by message-passing has its roots in operating system architectures studied in the 1960s [5]. The first widespread distributed systems were local-area networks such as Ethernet, which was invented in the 1970s [6]. E-mail was the most successful application of ARPANET [45][39] and it is probably the earliest example of a large-scale distributed application. In addition to ARPANET, its successor, the Internet, and other early worldwide computer networks like Usenet and FidoNet from 1980s, were used to support distributed discussion systems.

The SAPO Computer built in Prague, Czechoslovakia was probably the first Fault-Tolerant Computer which was built in 1950-1954 under the supervision of Antonin Svoboda. It used relays and a magnetic drum and was operated during 1957-1960.

John H. Wensley et al [1978] designed an ultrareliable Computer for critical aircraft control application that achieves fault tolerance by replication of tasks among processing units called SIFT (Software Implemented Fault Tolerance). The main processing units are off-the-shelf minicomputers, with standard microcomputers serving as the interface to the I/O system. Fault isolation is achieved by using a specially designed redundant bus system to interconnect the processing units. Error detection and analysis and system reconfiguration are performed by software. Iterative tasks are redundantly executed, and the result of each iteration is voted upon before being used. Thus, any single failure in a processing unit or bus can be tolerated with triplication of tasks, and subsequent failures can be tolerated after reconfiguration [26].

Hermann Kopetz et al [1989] in their paper named ‘distributed fault-tolerant in real-time system: The Mars approach’ discussed an approach which achieves fault tolerance by means of active redundancy (logically by sending each message twice and physically by having two or more components execute the same tasks). The Mars dependability analysis determines the degree of redundancy required. In this approach each component has intrinsic self checking mechanisms that safeguard the component’s fail-stop behaviour for fault isolation which results in a condition where a component either sends correct information or it remains silent [23].

Bagchi and Hakimi [1991] presented an algorithm for diagnosing faulty processors in arbitrary networks. According to their design, each fault free processor knows only about itself and its physical neighbours initially. Then the fault-free processors start the algorithm by walking up and initiating the formation of a tree based testing topology. Multiple trees that are being formed simultaneously are merged into one. Diagnosis information is sent along with the new message that forms the tree.

Lui Sha and Shirish S. Sathaye [1993] described the use of generalized rate monotonic scheduling theory for the design and analysis of a distributed real-time system [29].

Christian [1993] said that before discussing fault tolerance, we need to design the building blocks and the failure these blocks would experience. So, the design of a fault tolerant system would be easier and effective in any fault environment [13].

Hermann Kopetz and Gunter Grunsteidl [1994] designed a protocol named Time-Triggered Protocol (TTP) which is an integrated communication protocol for time-triggered architectures. It provides the services required for the implementation of a fault-tolerant real-time system: predictable message transmission, message acknowledgment in group communication, clock synchronization, membership, rapid mode changes, and redundancy management. It implements these services without extra messages and with only a small overhead in the message size [24].

Fuxing Wang and Krithi Ramamritham [1995] presented an approach that mathematically determines the replication factor for tasks. Their approach is a task schedule based analysis rather than a state based analysis as found in other models and therefore could provide a fast method to determine optimal redundancy levels. This method is not limited to hardware reliability given by constant failure rate functions as in most other models [19].

In 1997, Andre Schiper said that a system failure occurs when the system behaviour is not consistent with its specification. According to him, “more the number of components more are the things that could be faulty”.

Eltefaat Shokri et al [1997] made an effort on the design and implementation of an adaptive fault-tolerance middleware (AFTM) using a CORBA-compliant object request broker resting on the Solaris open system platform and also briefly discussed the essential capabilities of AFTM, the overall system architecture, and its design decisions in their paper [18].

Felix Gartner [1999] mentioned that the key to Fault Tolerance is redundancy and no matter how well designed or how fault tolerant a system is, there is always the possibility of failure if the faults are too frequent or too severe. According to his observation, for a system to be able to tolerate faults, one must employ a form of redundancy.

For removing a fault/failure from the system it has to be detected first and then fault tolerance technique can be applied. Many failure detections have been explained in various papers as an independent service [22] [42] [21] [24].

Replication, is the process of sharing information by which it can ensure consistency between redundant resources (i.e. software or hardware components), to improve reliability, fault-tolerance, or accessibility. Job replication is the method of replicating job on multiple server such as in grid computing service is capable of receiving jobs, executing them, performing checksum operations on them, and sending the result back to the client [3] [14] [27].

Data Replication is also commonly used by fault tolerance mechanisms to enhance availability in Grid like environments where failures are more likely to occur. In this, data is stored on multiple storage devices as a replica. Data replication may be synchronous or asynchronous depend upon data consistency. Components are replicated on different machines, and if any component or machine fail, then that application can be transferred and run on another machine having the required components [32] [39] [71].

M. Castro and Barbara Liskov [1999] presented a replication algorithm that can used to tolerate Byzantine Faults. This offers both liveliness and safety provided at most [(n-1)/3] out of the total of n-replicas that are simultaneously faulty. This means that Clients eventually receive replies to their request and these replies are correct according to linearizability. The algorithm incorporates important optimization that enables it to perform efficiently and it works in synchronous system like the Internet.

Louca et al [2000] in their paper named ‘MPI-FT: portable fault tolerance for MPI’ explained the technique known as online scheduling in MPI-FT to tolerate fault in MPI [33]. In this authors propose the mechanism for detecting process failure and there recovery mechanism in case of failure. In monitoring process, each process keeps a buffer with its own message traffic to be used in case of failure and the implementer uses periodical test for notification of failure. For recovery all the communications of processes are simulating with dead one and solution is provided for dead process.

Kjetil Nørvag [2000] in his paper ‘An introduction to fault-tolerance systems’, mainly from the hardware point of view, studied the different ways of achieving fault-tolerance with redundancy. Finally, he studied some systems as case examples, including Tandem, Stratus, MARS, and Sun Netra ft 1800 and concluded with a statement that “Fault-tolerance techniques will become even more important the coming years” [28].

According to Ghosh (2007), Lynch (1996), Peleg (2000), in a distributed environment, the system has to tolerate failures in individual computers [20][34][41].

Paul Stellin et al [1999] propose a fault detection service which was designed to be incorporated in a modular fashion, into distributed computing systems, tools, or applications. This service uses well-known techniques based on unreliable fault detectors to detect and report component failure, while allowing the user to tradeoff timeliness of reporting against false positive rates. They also described the architecture of this service report on experimental results that quantify its cost and accuracy, and its use in two applications, monitoring the status of system components of the GUSTO computational grid testbed and as part of the NetSolve network-enabled numerical solver [40].

Greg Bronevetsky et al [2003] designed a technique for Automated Application-level Checkpointing of MPI Programs. It is the process to saving from complete execution of a task. It checks the acceptance test, if fail then go to previous checkpoint instead of beginning. A check point may be system level, application level, or mixed level depends on its characteristics [22].

Xavier D´efago et al [Sept. 2003] in their paper on the Design of a Failure Detection Service for Large-Scale Distributed Systems, highlighted the main issues related to ensuring failure detection in large-scale systems, and overviewed the main solutions proposed in the literature so earlier. They outlined a pragmatic architecture for a failure detector service based on the failure detector, and a combination of techniques proposed in related work [48].

Paul Townend and Jie Xu [2003] in their paper “Fault tolerance within a grid Environment” showed that fault tolerance can be achieved in loosely coupled job scheduling with job replication scheme, such that, jobs that are efficiently and reliably executed can be tolerated [37].

Paul Townend et al [2004] proposed a replication-based fault tolerance scheme that allows voting to be performed on results of functionally-equivalent services, with the aim of detecting any incorrect results whilst bringing about potential improvements in performance [38].

Bernhard Fechner and Jörg Keller [2004] presented a voting scheme for multithreaded environments which is based on the observation that a fault tolerant system which does not know its history cannot distinguish between transient and permanent errors, caused by use of a faulty component. In contrary to simpler voting schemes it is able to recognize and distinguish transient and permanent faults. The history of errors is used to predict future errors and to determine if a permanent or transient error occurred [8].

Alain Girault et al [2004] designed an algorithm which is an active replication scheme that tolerates failure in distributed embedded real time System. Processor and communication link failure can be tolerated by using offline scheduling technique and generate a fault tolerate distributed schedule. In the proposed method, graph transformation is used to perform software redundancy, where a given input algorithm graph (Alg) is transformed into a new algorithm graph (Alg*) augmented with redundancies. Then, operations and data-dependences of Alg* can be distributed and scheduled on a specified target distributed architecture (Arc) to generate a fault tolerant distributed schedule. This algorithm is based upon graph transformation can tolerate fixed number of arbitrary processor and communication link failure [7].

L. P. Saikia and K. Hemachandran [2007] discussed the problem of distributed diagnosis in arbitrary network failures and repairs. In a distributed system, fault tolerance capability can be incorporated by providing the system with redundant resources. The purpose of their study was to simulate a distributed system and carry out fault diagnosis under Arbitrary Network topologies. In this algorithm, nodes detect failure in neighbouring nodes and then propagate this information to other nodes in the network in two discrete steps: first, detection and second, dissemination. The failure information propagated by the nodes consists of failure events, that is, a transition of a node from *fault-free to faulty or faulty to fault-free condition* [30].

TERCOS [2007] [46] and DEBUS [2009] are two of the best approaches used for exploiting redundancies in Fault-Tolerant and Real-Time Distributed Systems [47]. These techniques are based upon offline scheduling for exploiting redundancy in which Tercos can significantly improve schedulability by up to 17.0% (with an average of 9.7, whereas Debus can enhance schedulability over Tercos by up to 12% (with an average of 7.8%).

Negar Mosharraf and Mohammad Reza Khayyambashi [2009], in their paper, proposed an FT-CORBA structure as a structure used for supporting fault tolerance programs as well as relative important parameters including replication style and number of replica which play further role in improved performance and making it adaptive to real time distributed system have been reviewed. Studying these specifications have been made a structure adaptive to real time systems with higher performance than FT-CORBA and, finally the implementing of the said structure and determination of the number of replica and the replication style as well as the significance of related parameters have been investigated [36].

Arvind Kumar et al [2011], in their paper discussed the different techniques of fault tolerance. Their study focused on the types of faults, fault detection and recovery techniques. Based on how a system behave after failure, the fault tolerant systems can be classified in three different types: (i)Fail Stop System, (ii)Byzantine System and (iii)Fail-Fast System. They have mentioned some approaches for fault tolerance in Real Time distributed system. These are:

a) *Replication*
    - Job Replication
    - Component Replication
    - Data Replication
b) *Check pointing*
c) *Scheduling/Redundancy*
    - Space Scheduling/Redundancy
    - Time Scheduling/ Redundancy.
    - Hybrid Redundancy [2]

Sheheryar Malik and Fabrice Huet [2011], proposed a fault tolerance mechanism for real time computing on cloud infrastructure. It has all the advantages of forward recovery mechanism. It has a dynamic behaviour of reliability configuration. The scheme is highly fault tolerant. The reason behind adaptive reliability is that the scheme can take advantage of dynamic scalability of cloud infrastructure [44].

Ban M. Khammas [2012], designed a fault tolerance system for soft real time distributed system (FTRTDS). This system was designed to be independent of specific mechanism and facilities of the underlying real time distributed system. The research was done in a specially chosen environment. The FTRTDS has a distributed unit and a central unit. The Distributed Fault Tolerance Unit (DFTU) was distributed among all computers in the system and it ran with the system boot of that computer. The Central Fault Tolerance Unit (CFTU) runs manually by the user in one computer chosen for. It needs backup software for each part of distributed system to use it when necessary [9].

Lakshmi Prasad Saikia et al [2013] presented a paper giving a detailed literature survey in Fault Tolerance for Distributed Computing Systems and thereby discussing about the faults, error and failure that a distributed system can face [31].

Shyam Bhati et al [2013], proposed a fault tolerance mechanism where all dispatchers handle client requests assigned to them and exchange heartbeat messages [35] for failure detection. They also evaluated the fault-tolerant distributed Web systems with multiple dispatchers, which are registered in the local authoritative name server [43].

Lakshmi Prasad Saikia and Kundal Kr Medhi [2013] in their paper discussed the different faults and distributed fault tolerant schemes and approaches in real time environment [32].

Bibhudatta Sahoo, Aser Avinash Ekka [2007], said that Real-time distributed system, which is designed to provide solutions in a stringent timing constraint requires fault-tolerance. Their paper presented a new fault-tolerant scheme and an adaptive FT on heterogeneous multi-component distributed system architecture based on Distributed Recovery Block (DRB). The experiment showed that, the proposed scheme based on DRB with Random-EDF heuristic acting upon periodic tasks with timing and precedence constraints can tolerates about 10% to 20% number of permanent failures and an arbitrary number of timing failures [10][12][18].

C. M. Krishna [2014], surveyed the problem of how to schedule tasks in such a way that deadlines continue to be met despite processor (permanent or transient) or software failure. In the survey he found that since the reliability of a real-time system is related to the probability of meeting its hard deadlines, these reliability requirements translate to the need to meet critical task deadlines with a very high probability [11].

Abhilash Thekkilakattil et al [2014], presented a method for scheduling mixed criticality real-time tasks on a distributed platform in a fault tolerant manner while taking into account the recommendations given by the reliability studies like ZHA and FHA. First, they used mathematical optimization to allocate tasks on the processors, and then derive fault tolerant and fault aware feasibility windows for the critical and non-critical tasks respectively. Finally, they derived scheduler specific task attributes like priorities for the fixed priority scheduler. Their method provides hard real-time fault tolerance guarantees for critical tasks while maximizing resource utilization for non-critical tasks [2].

Eric Missimer et al [2014] in their paper Distributed Real-Time Fault Tolerance on a Virtualized Multi-Core System presented different approach for real-time fault tolerance using redundancy methods for multi-core systems. Using hardware virtualization, a distributed system on a chip is created, where the cores are isolated from one another except through explicit communication channels. Using this system architecture, redundant tasks that would typically be run on separate processors can be consolidated onto a single multi- core processor while still maintaining high confidence of system reliability [17].

IV. CONCLUSION

We have studied various Fault Tolerance techniques used in distributed real time environment. Since the development of the Distributed systems, fault tolerance has always been a very critical issue. Some of the recent works includes approaches for real-time fault tolerance using redundancy methods for multi-core systems developed for hardware virtualization, that is, a distributed system on a chip which uses new recovery techniques that require higher bandwidths and lower latencies than those of traditional networks. Another work is an approach to fault tolerance called bounded-time recovery (BTR), intended for systems that need strong timeliness guarantees during normal operation but can tolerate short outages in an emergency cyber-physical systems. After studying and scrutinizing a wide range of literature, it has become clear that, the issue of fault tolerance in real-time constraints for distributed system is a real problem worth researching into.

REFERENCES

[1] Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11-33, January 2004.

[2] Abhilash Thekkilakattil, Radu Dobrin, Sasikumar Punnekkat, *Mixed Criticality Scheduling in Fault-Tolerant Distributed Real-time Systems*, International Conference on Embedded Systems, 2014.

[3] Arvind Kumar, Rama Shankar Yadav, Ranvijay, Anjali Jain, *Fault Tolerance in Real Time Distributed System,* International Journal on Computer Science and Engineering (IJCSE), Vol. 3 No. 2 Feb 2011.

[4] Alain Girault, Christophe Lavarenne, Mihaela Sighireanu, Yves Sorel. "*Generation of Fault-Tolerant Static Scheduling for Real-Time Distributed Embedded Systems with Multi-Point Links".* In IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, San Francisco, USA, April 2001.

[5] Andrews, Gregory R., *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison–Wesley, ISBN 0-201-35752-6, p. 348. 2000.

[6] Andrews, Gregory R., *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison–Wesley, ISBN 0-201-35752-6, p. 32, 2000.

[7] Alain Girault, Hamoudi Kalla, and Yves Sorel "*An active replication scheme that tolerates failure in distributed embedded real time system*" IFIP 18th World Computer Congress TC10 Working Conference on Distributed and Parallel Embedded Systems (DIPES 2004) 22–27, Toulouse, France, August 2004.

[8] Bernhard Fechner, Jörg Keller, *A fault-tolerant voting scheme for multithreaded environments*, Parallel Computing in Electrical Engineering, 2004.

[9] Ban M. Khammas, "*Design of Fault Tolerance in Real Time Distributed System*", "Vol.8, No.1, PP11-17", "Al-Khwarizmi Engineering Journal", 2012.

[10] Bibhudatta Sahoo, Aser Avinash Ekka, "*Backward Fault Recovery in Real Time Distributed Systems of Periodic Tasks with Timing And Precedence Constraint*", Proceedings of the International Conference on Emerging Trends in High Performance Architecture, Algorithms and Computing, July 01/2007.

[11] C. M. Krishna, *Fault-tolerant scheduling in homogeneous real-time systems*, ACM Computing Surveys (CSUR) Volume 46 Issue 4, Article No. 48 , ACM, New York, NY, USA, April 2014.

[12] C. Fetzer, M. Raynal, and F. Tronel. *An adaptive failure detection protocol*. In Proc. 8th IEEE Pacific Rim Symp. On Dependable Computing (PRDC-8), pages 146–153, Seoul, Korea, Dec. 2001.

[13] Christian Flaviu, *"Understanding Fault Tolerant Distributed System",* May 1993.

[14] Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair (2011*). Distributed Systems: Concepts and Design (5th Edition)*. Boston: Addison-Wesley. ISBN 0-132-14301-1. 2011.

[15] Dima, Alain Girault, Christophe Lavarenne, Yves Sorel, *Off-line real-time fault-tolerant scheduling.* In Euromicro Workshop on Parallel and Distributed Processing, Mantova, Italy, February 2001.

[16] Dolev, Shlomi, "*Self-Stabilization"*, MIT Press, ISBN 0-262-04178-2, 2000.

[17] Eric Missimer, Richard West, Ye Li, *Distributed Real-Time Fault Tolerance on a Virtualized Multi-Core System*, OSPERT 2014, 2014.

[18] Eltefaat Shokri, Herbert Hecht, Patrick Crane, Jerry Dussault, K. H. (Kane) Kim, *An Approach for Adaptive Fault-Tolerance in Object-Oriented Open Distributed Systems*, feb 1997.

[19] Fuxing Wang and Krithi Ramamritham, *Determining Redundancy Levels for Fault Tolerant Real-Time Systems,* IEEE transactions on computers. Vol. 44, no. 2, February 1995.

[20] Ghosh, Sukumar, *Distributed Systems – An Algorithmic Approach*, Chapman and Hall/CRC, ISBN 978-1-58488-564-1, p. 192–193, 2007.

[21] Gosia Wrzesin´ ska, Rob V. van Nieuwpoort, Jason Maassen, Henri E. Bal, *Fault-tolerance, malleability and migration for divide-and-conquer applications on the grid*, in: Proceedings of International Parallel and Distributed Processing Sympo- sium .IEEE, 2005.

[22] Greg Bronevetsky, Daniel Marques, Keshav Pingali, Paul Stodghill, *Automated Application-level Checkpointing of MPI Programs*, Cornell University, Ithaca, NY, 2003, pp. 84.94.

[23] Hermann Kopetz, Andreas Dam, Christian Koza, Marco Mulazzani, Wolfgang Schwabi, Christoph Senft, Ralph Zainlinger, *Distributed fault-tolerant real-time system: the mars approach,* February 1989.

[24] Hermann Kopetz and Gunter Grunsteidl, '*TTP: Time Triggered protocol for Fault-Tolerant Real-Time Systems'*, IEEE 1994.

[25] Insup Lee, *Introduction to the Distributed Real-Time System, CIS 541, Spring 2010.*

[26] John H. Wensley, Leslie Lamport, Jack Goldberg Milton W. Green, Karl N. Levi'it, P. M. Melliar-Smith, Robert E. Shostak And Charles B. Weinstock, *Sift: Design And Analysis of a Fault-Tolerant Computer for Aircraft Control*,1978.

[27] J .Coenen, J. Hooman, "*A Formal Approach to Fault Tolerance in Distributed Real-Time Systems"*, Department of Mathematics and Computing Science, Eindhoven University of Technology, Netherland, 1990.

[28] Kjetil Nørv°ag, *An Introduction to Fault-Tolerant Systems*, Department of Computer and Information Science, Norwegian University of Science and Technology, 7034 Trondheim, Norway, July 2000.

[29] Lui Sha,Shirish S. Sathaye, *Distributed Real-Time System Design:Theoretical Concepts And Applications*, Technical Report CMU/SEI-93-TR-002, ESC-TR-93-179, March 1993.

[30] Lakshmi Prasad Saikia and K. Hemachandran, "*Simulation of System Level Diagnosis in Distributed Arbitrary Network*", "Journal of Theoritical and Applied Information Technology", 2007.

[31] Lakhmi P. Saikia , Nilotpal Baruah, and K.Hemachandran, "*System Diagnosis and Fault Tolerance for Distributed Computing system*", "International Journal for Computer Science and Communication Networks" Vol.3, Issue 5, 284-295, Oct 2013.

[32] Lakshmi Prasad Saikia and Kundal Kr. Medhi "Distributed Fault Tolerance System in Real Time Environments", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 11, November 2013.

[33] Louca, neophytou. Lachanas. And evripidou "*MPI-FT: portable fault tolerance for MPI" In parallel processing latters* Vol 10,no. 4 pg.371-382. 2000.

[34] Lynch, Nancy A., *Distributed Algorithms*, Morgan Kaufmann, ISBN 1-55860-348-4, p. 2–3, p. 99–102, 1996.

[35] M.K. Aguilera, W. Chen, S. Toueg, "Heartbeat: *A time- out free failure detector for quiescent reliable communications",* in: Proceedings of the 11th International Workshop on Distributed Algorithms, WDAG.97, pp. 126.140. , September 1997.

[36] Negar Mosharraf, Mohammad Reza Khayyambashi*, "Improving performance and reliability of adaptive fault tolerance structure in distributed real time systems",* proceedings of 9th WSEAS International Conference on Applied Informatics and Communications, 2009.

[37] Paul Townend, Jie Xu," *Fault tolerance within a grid Environment*", As part of the e-Demand project at the University of Durham, DH1 3LE, United Kingdom, 2003.

[38] Paul Townend, Jie Xu, "*Replication-based fault tolerance in a grid environment"*, As part of the e-Demand project at University of Leeds, Leeds, LS2 9JT, UK, 2004.

[39] Peter , "*The history of email"*, Internet History Project, 2004.

[40] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. "*A fault detection service for wide area distributed computations".*Cluster Computing,Volume 2, Issue 2, pp 117-128, July 1999.

[41] Peleg, David, "*Distributed Computing: A Locality-Sensitive Approach"*, SIAM, ISBN 0-89871-464-8, p. 4, 2000.

[42] R. van Renesse, Y. Minsky, and M. Hayden. "A *gossip-style failure detection service.* In N. Davies, K. Raymond, and J. Seitz, editors, Middleware 1998, pages 55-70, The Lake District, UK, 1998.

[43] Shyam Bhati, Sandeep Sharma, Karan Singh, "*A Fault-Tolerant Based Load Balancing Model in Distributed Web Environment"*, May. 2013.

[44] Sheheryar Malik and Fabrice Huet, *"Adaptive Fault Tolerance in Real Time Cloud Computing"*, IEEE World Congress on Services, Nov 2011.

[45] T. D. Chandra and S. Toueg. "*Unreliable failure detectors for reliable distributed systems.*" Journal of the ACM, 43(2): 225-267, Mar 1996.

[46] Wei Luo, FuMin Yang, Gang Tu, LiPing Pang, Xiao Qin *"TERCOS: A Novel Technique for Exploiting redundancies in Fault-Tolerant and Real-Time Distributed Systems"* 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications(RTCSA 2007).

[47] Wei Luo, Xiao Qin, Xian-Chun Tan, Ke Qin, and Adam Manzanares "*Exploiting Redundancies to enhance Schedulability in Fault-Tolerant and Real-Time Distributed Systems"* IEEE Transactions on system man and cybernetics – part A : systems and humans, VOL. 39, NO. 3, May 2009.

[48] Xavier D´efago, Naohiro Hayashibara, Takuya Katayama, *On the Design of a Failure Detection Service for Large-Scale Distributed Systems*, Appeared in Proc. Intl. Symp. Towards Peta-Bit Ultra-Networks (PBit 2003), pp.88–95, Ishikawa, Japan, Sept. 2003.