Proceedings of the International Conference on
Emerging Trends in High Performance Architecture, Algorithms & Computing (HiPAAC 2007)
(2007) 124-130

Backward Fault Recovery in Real Time Distributed Systems of Periodic Tasks with Timing And Precedence Constraint

Bibhudatta Sahoo
Department of CSE, NIT Rourkela
bdsahu@nitrkl.ac.in

Aser Avinash Ekka
Department of CSE, NIT Rourkela
avinash.ekka@nitrkl.ac.in

Abstract

Real-time distributed system, which is designed to provide solutions in a stringent timing constraint requires fault-tolerance. This paper presents a new fault-tolerant scheme and an adaptive FT on heterogeneous multi-component distributed system architecture based on Distributed Recovery Block (DRB). The experiment shows 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.

1 Introduction

The computational environments dealt by Distributed heterogeneous computing consists of multiple heterogeneous computing modules, these modules interact with each other to solve a problem. Real-time Distributed systems (RTDS), such as aircrafts and automobiles, nuclear, robotics, and telecommunication, require high dependability, where system failures during execution can causes catastrophic damages. These systems must function with high availability even under hardware and software faults.

No matter how meticulously error avoidance and error detection techniques are used, it is virtually impossible to make a practical system entirely error free. Therefore, to achieve high reliability, even in situation where errors are present, the system should be able to tolerate the faults and compute the correct results this is called fault-tolerance. Fault-tolerance can be achieved by carefully incorporating redundancy.

The faults in a distributed computing system may appear either in the hardware or in the software and they can be classified as being permanent, intermittent or transient [6, 15, 5]. In fault-tolerant Real Time Distributed systems, detection of fault and its recovery should be executed in timely manner so that in spite of fault occurrences the intended output of real-time computations always take place on time. For fault tolerant technique detection, latency and recovery time are important performance metrics because they contribute to node downtime. A fault tolerant technique can be useful, in RTDS if its fault detection latency and recovery time are tightly bounded. When this is not feasible, the system must attempt the fault tolerance actions that leads to the least damage to the applications mission and the systems users. One major advantage of distributed systems is to tolerate individual component failure without terminating the entire computation [8, 9, 10]. Research in fault-tolerant distributed computing, aims at making distributed systems more reliable by handling faults in complex computing environments. Moreover, the increasing dependence of different services on real time heterogeneous distributed system has led to an increasing demand for dependable systems, systems with quantifiable reliability properties. Task scheduling techniques can be used to achieve effective fault tolerance in real time systems [10, 8]. This is an effective technique, as it requires very little redundant hardware resources. Fault tolerance can be achieved by scheduling additional ghost copies in addition to the primary copy of the task. we present our approach based on software redundancy to tolerate permanent and timing failures. We propose to use distributed recovery block to perform software redundancy, where a given input a task (T_i) is augmented with a redundancies. Then, operations and data-dependences of T_i can be distributed and scheduled on a specified target distributed architecture (G) to generate a fault tolerant distributed schedule.

2 Real Time Distributed Computing System

We consider a heterogeneous distributed computing system (HDCS) consists of a set W of n Nodes (uniquely addressable computing entity)$\{P_1, P_2, , P_n\}, P_i = (D_i, e_i)$, where D_i, is the set of tasks in the queue of P_i, e_i is the fixed execution rate. Each processor has different execution rate measured in MFLOPS/s[6] and they are connect with each other using bi-directional point-to-point communication links . In heterogeneous distributed system a task T_i has different computation time which is measured by which represents the time of task t_i on processor P_i where $1 \leq i \leq n$ and $1 \leq j \leq N$. The processors of the distributed system are heterogeneous and the availability of each processor can vary over time (processors are not dedicated can may have other tasks that partially use their resources). We have extended the model mentioned in [16] by adding each processor with a backup queue as shown in Figure 1 and in [17] we have shown that the performance of the proposed model for independent tasks increases the guarantee ratio.

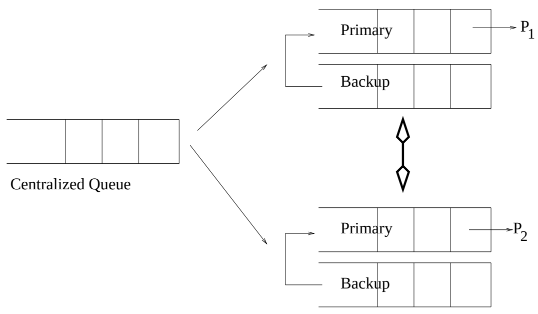


Figure 1. Example of FTRTDS architecture for 2 nodes

3 Periodic Task Model

A periodic task is characterized by its period of release and they are to be executed exactly once in every period. Period of the task may be its deadline. We denote the set of tasks in the application by the set $T = \{T_1, T_2, T_3 T_n\}$ where a task T_i is periodic. The basic task model is P modeled by a set of N periodic tasks:

$$\Pi = \{\tau_i = (P_i, D_i, C_i) \mid 1 \leq i \leq N\} \tag{1}$$

where

P_i is the period of the task. Each task is released every P_i time units. For non-periodic tasks, P_i is represented the minimum (or average) separation time between two consecutive releases. The difference in time between the arrivals of two consecutive instances of a periodic task is always fixed and is referred to as period of that task. Although the inter arrival times of instances of a periodic task are fixed, the inter release time may not be. D_i is the deadline, the period of time after the release time within which the task has to finish, Tasks can have arbitrary deadline. C_i is the worst-case execution time of the task at each release, i.e. the maximum time span between release of a task and end of the response of that release. In real time systems it is often necessary to determine an upper bound of time in that the program block is executed. We

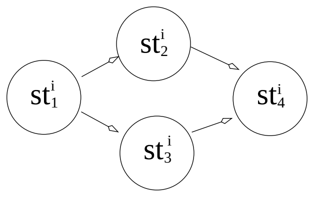


Figure 2. Task Graph for T_i with four subtasks

consider each Task T_i is assumed to consist of a set of subtasks, which execute "serially" along the level of the task graph. For convenience, we denote the set of subtasks of task $T_i = \{st^i_1, st^i_2, st^i_3, st^i_4\}$ as shown in Figure 3. where is the intermediate deadline [7].

We consider each Task T_i to consist of a set of subtasks, which have timing and precedence constraint [1] as shown in Figure 2. The subtasks st^i_4in a precedence constraint task T_i cannot be executed until the subtasks preceded by it have completed their execution i.e. st^i_2 and st^i_3. The subtasks of a task are considered to be the valid states of the given task.

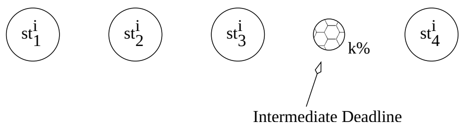


Figure 3. Subtasks and intermediate deadline

4 Previous Work

Occurrence of a fault during execution in any system requires, extra time to handle fault detection and recovery. In case of real time system in particular, it is essential that extra time be considered and accounted for prior to execution. The methods used for real time fault tolerant system, must consider the number and types of fault subjected to without violating the timing constraints. Fault tolerance has typically been approached from a hardware standpoint, with multiple replicas of essential applications running on separate hardware components mostly in parallel fashion. In the area of real time distributed systems, a fault-tolerant scheduling strategy is described in [18, 20, 21, 3]. The requirements of a fault tolerant scheduling algorithm in real time distributed systems are described in [7, 6]. A successful fault tolerant primary/backup algorithm with the dynamic EDF algorithm for multiprocessors running in parallel and executing real-time applications [6, 4]. One of the major technique for achieving fault tolerance is replication but the level of replication is chosen depending on the desired fault tolerance required [5] discusses a replication control mechanism in distributed real time database system. Software based fault tolerant application using a single version scheme (SVS) is described in [8, 13]. A middleware based MEAD infrastructure aims to provide a reusable, resource-aware real-time support to applications to protect against crash, communication, partitioning and timing faults are discussed in [2]. Kim also outlines the other middleware techniques of fault tolerance. Fault tolerant techniques implemented by means of scheduling are discussed in [11, 14, 22].

5 DRB Scheme

The Control Implementation Structures used in this paper is DRB: Distributed Recovery Block. In this paper we have outlined the two requirements for DRB variant i.e. Primary-Backup Fault tolerant algorithm in RTDS is that (1) the execution of backup versions should not hinder the execution of the primary version of the tasks, and (2) when the primary task fails to meet its deadline the backup instance should then be executed but it should be executed from the point of last correct subtask executed by the primary version. The first requirement is satisfied by not assigning the backup task to the processor for execution the second requirement is satisfied by the primary task communicating with the central scheduler and updates the backup to the last known successful state.

This paper we propose a new extended distributed recovery block based fault tolerant scheduling algorithm for real time tasks. Our algorithm ensures that the parallel updation of backup task works better in case of transient overload and handles both permanent and timing fault.

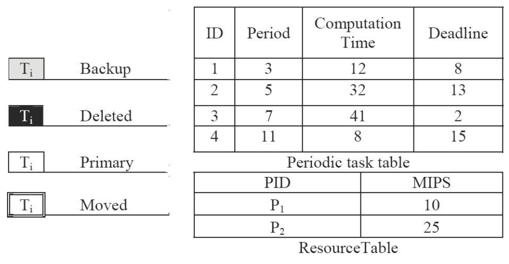
The distributed recovery block (DRB) scheme is an approach for realizing both hardware fault tolerance and software fault tolerance in real-time distributed and/or parallel computer systems. The underlying design philosophy behind the DRB scheme is that a real-time distributed or parallel computer system can take the desirable modular form of an interconnection of computing stations, where a computing station refers to a processing node (hardware and software) dedicated to the execution of one or a few application tasks [15]. The idea of the distributed recovery block (DRB) has been adapted from [7, 6]. Recovery block consists of one or more routines, called try blocks here, designed to compute the same or similar result, and an acceptance test which is an expression of the criterion for which the result can be accepted both in term of correctness and timing constraint. For the sake of simplicity a recovery block consists of only two try blocks, i.e. primary and backup [18, 4]. The error processing technique used is acceptance test that is parallel between node pairs but sequential in each node with complexity .

6 The Fault Tolerant Scheduling scheme

The main idea of software fault tolerance is to contain the damage caused by software faults. Several techniques that can be used to limit the impact of software faults (read bugs) on system performance. Efforts to attain software that can tolerate software design faults (programming errors) have made use of static and dynamic redundancy approaches similar to those used for hardware faults [8, 12]. Techniques involved in achieving software fault tolerance are: (i) timeouts, (ii) audits, (iii) exception handling, (iv) task rollback, (v) incremental reboot, (vi) voting, (vii) n-version programming, (viii) recovery-block approach, and (ix) algorithm based fault tolerance [19]

In fault-tolerant real time distributed systems, detection of fault and its recovery should be executed in timely manner so that in spite of fault occurrences the intended output of real-time computations always take place on time. For a fault tolerant technique detection latency and recovery time are important performance metrics because they contribute to server down-time. A fault tolerant technique can be useful, in RTDS if its fault detection latency and recovery time are tightly bounded. When this is not feasible, the system must attempt the fault tolerance actions that lead to the least damages to the application's mission and the system's users. We have proposed the following scheme that can be used to handle DRB based faults in RTDS. The algorithm makes sure that the backup tasks though scheduled to processors do not hamper the execution of primary tasks at the same time the backup task are updated according to the subtask completed in their primary counterpart so that when the primary task fails the backup task does not start its execution from beginning instead from the last updated subtask. When the primary task is completed within its deadline the backup task is terminated. The global picture of our methodology is shown in Figure 4- 7 and is given in Algorithm 1.

1. [Allocate resources to satisfy task deadline]
   Assign the primary and backup tasks to the distributed system in a RANDOMIZED fashion to different processors where the release time of both the primary and backup task is same.


Ti Backup
Ti Deleted
Ti Primary
Ti Moved

ID	Period	Computation Time	Deadline
1	3	12	8
2	5	32	13
3	7	41	2
4	11	8	15

Periodic task table

PID	MIPS
P_1	10
P_2	25

ResourceTable

Figure 4. Example of Task Parameters

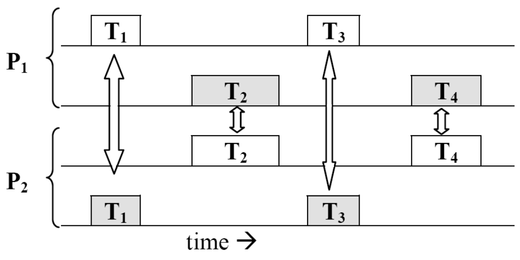


Figure 5. A possible initial schedule of given periodic tasks

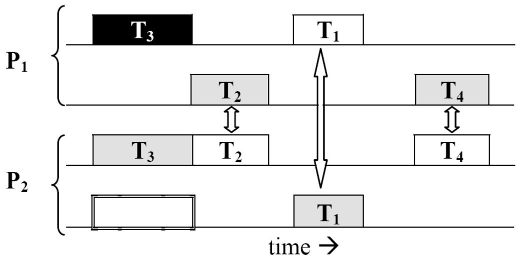


Figure 6. Schedule Update when T_3 misses its deadline At $Time = sysc_n$

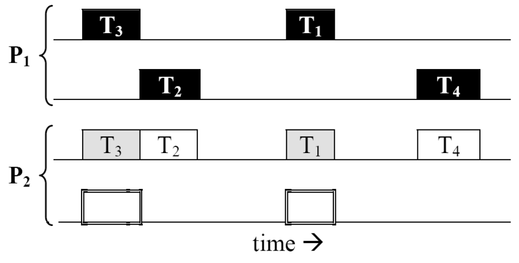


Figure 7. Schedule Update when P_1 encounters a crash fault At $Time = t_n$

2. Use the EDF algorithm as uniprocessor scheduling algorithm

3. Update the backup task according to the subtask covered by the primary task.

4. [Runtime monitoring of timing constraint]
   Check if a task misses its intermediate relative deadline with at least M% of the task is completed.

5. [Fault Tolerant Strategy]
   If the task misses its deadline then the primary task is terminated and the updated backup task at the scheduled processor is treated as the primary task.

6. If the backup task fails reject the task.

Algorithm 1. High level DRB based Fault Tolerant in RTDS

Algorithm 6.2 mentions our approach for fault tolerance in RTDS and Algorithm 6.1 mentions the fault injection algorithm. The results Figure 8- 9 show that our algorithm improves the performance, with regard to guarantee ratio

$$\begin{aligned} G &= System\ Guarantee\ Ratio \\ G &= \frac{Total\ Number\ of\ Tasks\ Guaranteed}{Total\ Number\ of\ Effective\ Tasks} \end{aligned} \tag{2}$$

of the traditional Faulty Random-EDF heuristic under permanent fault of 10% and 20% respectively and timing faults of tasks. We present a method that tolerates only permanent and timing failures. Adaptive FT Random-EDF is similar to Algorithm 6.2 except that when the Backup task replaces the Primary task one more replica is generated and scheduled as backup on different processor.

```
1. Input: a system resource set $G$
2. Select a processor $P_i$
3. IF No. of Faults in $(P_i) < N_{FP}$
   (a) Mark as FAULTY
   (b) Increment the Upper Limit
4. [End of if structure ]
```

Algorithm 6.1 Fault Injection Algorithm

```
1. Input: a set of periodic task set
   $T_i = \{st_1^i, st_2^i, st_3^i, st_4^i\}$ and a system resource set $G$
2. Repeat steps from Step 3 to Step 8 for
   $TIME = 1, 2, ...Sysc - 1, Sysc$
3. IF $period(T_k) = TIME$
        INSERT $T_k$ to the Central Scheduler Queue.
```

```
    [End of If structure ]

 4. IF Central Scheduler Queue $\neq$ NULL

        For each Primary version of task $T_i$
        Select the Non-Faulty end node $P_1$
        INSERT the Primary task to the
        $PrimaryQueue(P_1)$
        Create a Backup version of task $T_i$
        Select the Non-Faulty end node $P_2 \neq P_1$
        INSERT the Backup task to the
        $BackupQueue(P_2)$

    [End of If structure ]

 5. IF $period(FaultInjectionAlgorithm$
    $(SystemResourceSet)) = TIME$

        FAULT INJECTION ALGORITHM(
        System Resource Set)

    [End of If structure ]

 6. For each processor $P_i$ in the system

        IF new task added to the Primary Queue
            Rearrange the tasks in the Primary Queue
            of according to EDF.
            IF $Deadline(P_{T_i})$
            $< Deadline(PrimaryQueue[Front])$
                Preempt the currently executing task.
                Assign Primary Queue [Front] to the
                processor.
            [End of If structure ]
        [End of If structure ]

 7. Execute the task $T_i$ assigned to Processor from the
    $PrimaryQueue.$

 8. IF $IntermediateDeadline(T_i)$ exceeds TIME

        Terminate the primary version of task $T_i$
        Trigger a timing fault alarm
        Intimate the Backup version of task $T_i$ on
        remote processor $P_j$

    [End of If structure ]

 9. Update backup version of $T_i$ to the last valid
    subtask of the primary version of $T_i$

10. [Update the task from Backup Queue to Primary
    Queue if the primary task has failed ]

    Rearrange tasks in the Backup Queue whose
    primary task has failed according to EDF.

    DELETE Backup Queue [Front] and
    INSERT in Primary Queue

    Rearrange all the tasks in the
    Primary Queue according to EDF
```

Algorithm 6.2 Fault Tolerant Scheduling Algorithm

7 Experimental Analysis

To evaluate how well the proposed scheme performs, we compare the performance of Adaptive FT Random-EDF, FT Random-EDF with Faulty Random-EDF using a discrete event simulator developed by us using Matlab 6.0. The tasks are arriving into the systems dynamically in a periodic fashion, which are assigned to the processor by random selection, provided the processors memory is not full. The uniprocessor scheduler used is EDF algorithm. Timing fault are injected into the system when a task misses its deadline whereas crash or permanent fault are injected in random fashion with an upper an upper limit of 10

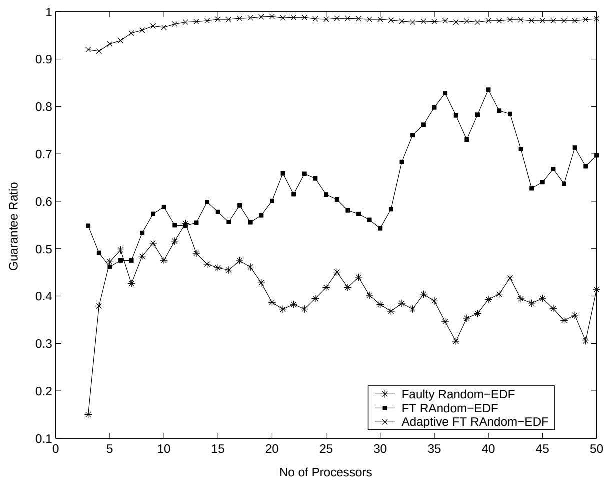


Figure 8. Guarantee Ratio of techniques in presence of $N_{FP} < 10\%$ and timing fault.

The results Figure 8- 9 show that our algorithm outperforms the traditional EDF uniprocessor scheduler, which has missed deadline in presence of timing and crash fault, and a randomized assignment of tasks.

8 Conclusion

Fault-tolerance becomes an important key to establish dependability in RTDS systems. Hardware and software redundancy are well-known effective methods for hardware

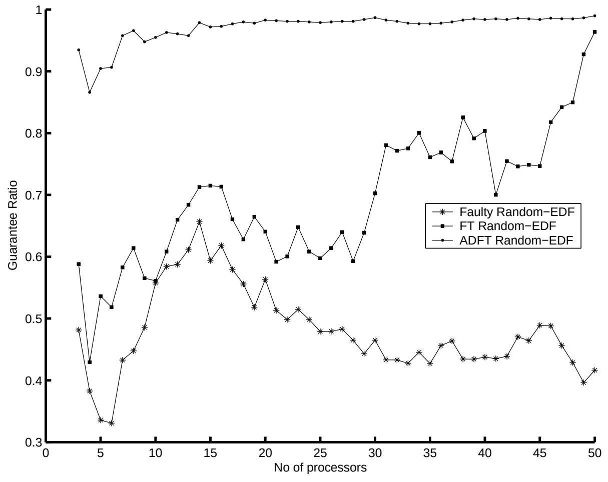


Figure 9. Guarantee Ratio of techniques in presence of $N_{FP} < 20\%$ and timing fault.

fault-tolerance, where extra hard ware (e.g., processors, communication links) and software (e.g., tasks, messages) are added into the system to deal with faults. We have investigated methods to overcome timing and permanent failures in heterogeneous real time distributed systems with point-to-point communication links. We have proposed a new method that tolerates at most N_{FP} permanent fault and arbitrary timing fault. This method is a software solution based on adaptive redundancy to overcome the failures. The percentage increase in the guarantee ratio of FT Random-EDF is 80% while the guarantee ratio of Adaptive FT Random-EDF heuristic is 99% with upto 10% of faulty systems for processor range in between 3 to 50. With upto 20% of faulty systems FT Random-EDF heuristic guarantee ratio is 70% while the guarantee ratio of Adaptive FT Random-EDF heuristic is 99%.

9 Acknowledgement

The work reported in this paper are being supported in part by R&D project grant 2005-2008 of MHRD Government of India with the title as Ëault Tolerant Real Time Dynamic Scheduling Algorithm For Heterogeneous Distributed Systemänd being carried out at department of Computer Science and Engineering, NIT Rourkela.

References

[1] Improving scheduling of tasks in a heterogeneous environment. *IEEE Transactions On Parallel And Distributed Systems*, 15(2), FEBRUARY 2004.

[2] M. S. A. Girault, C. Lavarenne and Y. Sorel. Generation of fault-tolerant static scheduling for real-time distributed embedded systems with multi-point links. In *Proceedings 15th International*, pages 1265 – 1272, April 2001.

[3] G. Attiya and Y. Hamam. Two phase algorithm for load balancing in heterogeneous distributed systems. In *Proceedings of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 434 – 439, Feb. 2004.

[4] G. M. I. Gupta and C. S. R. Murthy. *Primary-Backup based fault-tolerant dynamic scheduling of object-based tasks in multiprocessor real-time systems*, chapter 20. Dependable Network Computing. Kluwer Academic Publisher, MA, USA, 1999.

[5] H. K. Kim. Middleware of real-time object based fault-tolerant distributed computing systems: Issues and some approaches. In *Pacific Rim Int'l Symp. on Dependable Computing*, pages 3–8, December 2001.

[6] J. L. T. Kim, K.H. Goldberg and C. Subbaraman. The adaptable distributed recovery block scheme and a modular implementation model. In *Proceedings of Fault-Tolerant Systems*, pages 131 – 138, Dec. 1997.

[7] K. Kim. Designing fault tolerance capabilities into real-time distributed computer systems. In *Workshop on the Future Trends of Distributed Computing Systems*, pages 318 – 328. IEEE Proceedings, September 1990.

[8] K. Kim. Slow advances in fault-tolerant real-time distributed computing. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems*, pages 106 – 108, October 2004.

[9] C. Krishna and K. G. Shin. *Real Time Systems*. McGraw-Hill, 1997.

[10] R. Mall. *Real-Time Systems*. Pearson Education, 1st ed. edition, 2007.

[11] G. Manimaran and C. Murthy. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis. *IEEE Transactions on Parallel and Distributed Systems*, 9(11):1137 – 1152, November 1998.

[12] B. Mirle and A. M. K. Cheng. Simulation of fault-tolerant scheduling on real-time multiprocessor systems using primary backup overloading. Technical Report UH-CS-06-04, Real-Time Systems Laboratory, Department of Computer Science, University of Houston, May 2006.

[13] R. R. S. C. V. Raju and F. Jahanian. Monitoring timing constraints in distributed real-time systems. In *IEEE Real-Time Systems Symposium*, pages 57–67, 1992.

[14] R. M. S. Ghosh and D. Mosse. Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, pages 272–284.

[15] F. Z. S. H. Son and J.-H. Kang. Replication control for fault-tolerance in distributed real-time database systems. pages 73–81. IEEE, 1998.

[16] B. Sahoo and A. A. Ekka. Performance analysis of concurrent tasks scheduling schemes in a heterogeneous distributed computing system. In *Proceedings of the National Conference on Computer Science and Technology*, pages 11–21, KIET, Ghaziabad, November 2006.

[17] B. Sahoo and A. A. Ekka. A novel fault tolerant scheduling technique in real-time heterogeneous distributed systems using distributed recovery block. In *Proceedings of 3rd National Conference VISION07 on High Performance Computing*, pages 215–220, Tamil Nadu, INDIA, April 2007.

Government College of Engineering, Department of CSE, Tirunelveli.

[18] Y. K. T. Tsuchiya and T. Kikuno. Fault-tolerant scheduling algorithm for distributed real-time systems. In *Proceedings of the Third Workshop on Parallel and Distributed Real-Time Systems*, pages 99 –103, April 1995.

[19] A. Tyrrell. Recovery blocks and algorithm-based fault tolerance. pages 292 – 299. 22nd EUROMICRO Conference, Sept. 1996.

[20] H. J. L. P. X. Qin, Z. Han and S. Li. Real-time fault-tolerant scheduling in heterogeneous distributed systems. In *Proc. International Workshop Cluster Computing-Tech, Environments, and Applications*, pages 421–427, June 2000.

[21] H. Yongbing Zhang; Hakozaki, K.; Kameda and K. Shimizu. A performance comparison of adaptive and static load balancing in heterogeneous distributed systems. In *Proceedings of the 28th Annual Simulation Symposium*, pages 332 – 340, April 1995.

[22] Q. Zheng and K. G. Shin. Fault-tolerant real-time communication in distributed computing systems. *IEEE Transactions On Parallel And Distributed Systems*, 9(5), MAY 1998.