Design, Analysis and Implementation of Improved Adaptive Fault Tolerant Model for Cruise Control Multiprocessor System

Annam Swetha
Amrita Viswa Vidyapeetam
Coimbatore, India

Radhamani Pillay V
Amrita Viswa Vidyapeetam
Coimbatore, India

Sasikumar Punnekkat
Mälardalen University
Sweden

ABSTRACT
Safety or mission critical applications have to recover from an error within an acceptable time window or it may potentially lead to disastrous effects or higher costs. The usual industrial practice is to employ fault tolerance using hardware redundancy where costs are highly exorbitant depending on the mission. In this paper, we present a framework for adaptive fault tolerance on the commonly used hardware redundancy. This proposed model gives enhanced resource management and improved system performance under normal runtime and provides minimal safe functionality under error conditions. A new scheduling method, a combination of dynamic planning and dynamic best effort approach has been designed for joint scheduling of periodic and aperiodic tasks which also include online reconfiguration for error management. This fault recovery technique allows all critical tasks to meet their deadlines and the system continues functioning with minimal safe functionality upon errors. This model has been analyzed and evaluated on a practical case study of a Cruise Control System vis- à-vis a traditional redundancy scheme with simulation and validated with appropriate performance metrics. The results demonstrate the high performance throughput and process speedup (Execution time of process) that can be gained by applying this model to an m-processor redundancy model and the advantages can be accrued specially in the field of avionics in terms of fuel/weight ratio.

General Terms
Redundancy, Safety critical systems, real time scheduling

Keywords
Fault tolerance, resource management, Cruise control system, Process speedup.

1. INTRODUCTION
Safety critical systems are increasingly being employed in multiprocessor embedded platforms. The hardware redundancy traditionally employed for dependability increases the complexity and the cost of the system. As the level of redundancy goes up, indirectly this can lead to more faults and cost can become prohibitive. Under normal operation conditions the redundant units are not employed and hence the computing resources are being underutilized. These computing resources can be efficiently utilized and during fault condition dependability can be assured by the method explored in this paper.

Using task level criticality, where critical tasks have to meet their deadlines to avoid catastrophic effects, the proposed model ensures fault tolerance by scheduling the critical tasks on all the processors and sharing the noncritical tasks among the processors. By this approach the redundant units are efficiently utilized and there by the performance of the system is improved. The proposed scheduling scheme is denoted as Enhanced Resource Management Scheme (ERMS) and is evaluated in comparison with Traditional Redundancy Scheme (TRS). A ***Cruise Control System (CCS)*** which is one of the major safety critical unit in automotives is taken as a case study for the implementation of the proposed scheme and is validated with the suitable performance metrics. The main function of the CCS is to maintain a constant speed which is set by the driver there by reducing the work load of the driver. The rest of the paper is organized as follows. Section 2 references the literature survey and Section 3 gives the background study of scheduling of fault tolerant real time systems and the case study of the cruise control system. In section 4, the system model with TRS and proposed ERMS approach is discussed and the implementation of the case study has been detailed. In Section 5 the analysis and simulation results with performance evaluation is presented. The conclusion and future scope is given in Section 6.

2. RELATED WORKS
Multiprocessor real-time scheduling theory has it origins in the late 1970's. The seminal paper of Liu('78) [2] heavily influenced the course of research in this area for two decades. During the 1980's and 1990's partitioned approaches, with a fixed allocation of tasks to processors was preferred compared to global approach. In 1997, Phillips et al.[3] in his paper proposed the advantages of global scheduling which renewed interest in global scheduling algorithms.

J. Von Neumann [4], E. F. Moore, C. E. Shannon, [5] and their successors developed theories of using redundancy to build reliable logic structures from less reliable components, whose faults were masked by the presence of multiple redundant components. The theories of masking redundancy were unified by W. H. Pierce as the concept of failure tolerance in 1965 [6]. In 1967, A. Avizienis integrated masking with the practical techniques of error detection, fault diagnosis, and recovery into the concept of fault-tolerant systems [7]. Further a fault tolerant scheduling algorithm for multiprocessors was analyzed by Ghosh [8]. Krishna and Shin [9] proposed a fault tolerant scheme for quick recovery of tasks from failure. Oh and Son.[10] proposed a scheme that enhances the fault tolerance in static realtime scheduling.

Later he proposed the concept of online scheduling of multiple versions of tasks on the minimum number of processors under the RM scheduling policy[11]. Manimaran [12] has proposed dynamic algorithms to schedule real-time tasks on multiprocessors by employing the primary backup fault-tolerance strategy. Mahmud Pathan[13] presented a new scheduling algorithm that integrates timeliness and criticality to fault tolerance. In[14, 15,16] the authors have proposed an improved resource managements technique with an innovative fault tolerant paradigm. This technique enhances the performance of the system and effectively utilizes the additional resources. One of the applications of safety critical systems is Cruise Control System, a prototype of which has been developed in the year 1987, as part of an initiative of the European Union EUREKA program[17]. Its main objective is to reduce the driver workload and make the drive more comfortable by automatically adjusting the speed of the vehicle with regard to the surroundings.

3. BACKGROUND STUDY

Multiprocessor scheduling consists of global scheduling and partitioned scheduling where the former has a global scheduler, scheduling all the tasks to the available processors and in the latter tasks are pre-allocated to the processors[3]. The local scheduler in each processor determines the schedule for each processor using an uniprocessor scheduling policy.

Uniprocessor scheduling consists of offline scheduling and online scheduling where in former case a complete knowledge of the task set and time attributes are known and scheduling decisions are pre-computed offline and in latter case scheduling decisions are made during runtime, Audsley[18]. Online scheduling is flexible and adaptive but incurs significant overheads.

3.1 Scheduling Paradigms:

Static table-driven approach-These perform static schedulability analysis and the resulting schedule (or table, as it is usually called) is used at run time to decide *when* a task must begin execution.

Static priority driven preemptive approach - These perform static schedulability analysis but unlike in the previous approach, no explicit schedule is constructed. At run time, tasks are executed with “highest priority first” principle.

Dynamic planning-based approach - Unlike the previous two approaches, feasibility is checked at run time, i.e., a dynamically arriving task is accepted for execution only if it found feasible. One of the results of the feasibility analysis is a schedule or plan that is used to decide when a task can begin execution.

Dynamic best effort approach - Here no feasibility checking is done. The system tries to do its best to meet deadlines. But since no guarantees are provided, a task may be aborted during its execution [19].

From the above approaches following are selected for implementation in this work

Static table-driven approach - Periodic task scheduling during run-time normal mode. Given task characteristics, a table is constructed, that identifies the start and completion time of each task and tasks are dispatched according to this table.

Dynamic online best planning approach - Mixed Task scheduling during normal run-time mode. It provides the flexibility of dynamic planning approach with the predictability of best effort that checks for feasibility. In this approach after the arrival of aperiodic task, an attempt is made to create a schedule that contains the previously guaranteed tasks as well as the aperiodic task.

3.2 Fault tolerance

Fault tolerance is achieved through redundancy, Avizienis [7]. Some types of redundancy are hardware redundancy, based on replication of physical components, software redundancy, which provides different software versions of tasks, preferably written independently. Time Redundancy based on multiple executions of a task on the same hardware in different instances of time. Two general approaches proposed for hardware fault recovery are fault masking and dynamic recovery. Fault masking is a structural redundancy technique that completely masks the faults with in a set or redundant modules, their outputs are voted to remove the errors caused by the faulty module [20]. Triple Modular redundancy (TMR) is a commonly used form of fault masking in which the circuitry is triplicated and voted. Dynamic recovery is a technique used when only one copy of computation is running at a time and it involves automated self repair. In this case special mechanisms are required to detect faults in the modules, switch out a faulty module and switch in a spare module.

3.3 Safety critical systems

Safety critical systems are hard real time systems where missing deadlines of critical tasks results in catastrophic effects, Knight [21]. With the advent of increased development in on-chip technology multiprocessors came into existence for highly complex and sophisticated systems like safety critical systems. Well known examples of such systems include medical devices, avionics, aircraft flight control system, and nuclear systems.

Tasks in safety critical systems Periodic tasks are time driven tasks which occur at regular intervals of time, example - task which monitors the temperature of the patient in a patient monitoring system. Aperiodic tasks are event driven tasks which occur due to dynamic changes in the environment or they can be user initiated tasks, example - task that is activated on detecting an abnormality in the condition of the patient.

Task set consists of a set of tasks in an application classified as critical, non-critical and optional tasks based on the criticality level of the task. In some sense, non-critical and optional tasks can be considered as soft real time tasks. In general, all *controlling* and *actuating* tasks are considered as critical tasks and *sensing* tasks are considered as non-critical tasks. One such critical task is the task in the patient monitoring system which controls the oxygen supply to the patient in ICU.

Task graph Task set is transformed in to task graph where tasks are related by dependency. The generated graph consists of set of nodes corresponding to tasks and a set of edges corresponding to task dependencies.

A new way of fault tolerant scheduling in safety critical multiprocessors is explained in detail in [14, 15, 16], where the task level criticality has been explored to schedule the tasks in multiple processors. This ensures the safe functionality of the system even during the occurrence of fault by operating in different modes where certain optional tasks and non-critical tasks are dropped.

3.4 Performance metrics

In order to measure the performance of the fault tolerant scheme, the following performance metrics are chosen.

3.4.1 Effective Utilization U_e

It is the normalized utilization of the processors during the execution of an application Ramamritham[22].

3.4.2 Process Speedup S_p

Process Speedup indicates the overall execution time for the process, Davis [23].

3.4.3 Guarantee ratio G_r

It is the ratio of arriving aperiodic tasks that can effectively meet their deadlines to the number of tasks that are scheduled on the processor, Manimaran[24].

3.4.4 Average response time R_a

It is the average response time of soft aperiodic tasks that are scheduled on the processors, Sprunt [25].

3.4.5 Deadline miss ratio D_r
It is the ratio of tasks that has missed the deadline to the total number of tasks in the system, Chenyang[26].

3.5 Case study: Cruise Control System
The Cruise Control System (CCS) is characterized as an embedded real time system having a number of processors, sensors and actuators. The functionality of the system is based on assuming different classes or tasks that interact among each other in real time, Staines[27]. The tasks identified in the cruise control system can be categorized as periodic and aperiodic (user initiated).

i. Periodic tasks – all sensing actions
ii. User initiated tasks – Brake application and acceleration

3.5.1 Basic tasks
- Sensors scan processes (GPS, User Interface (UI), Brake, Accelerator, Engine)
- Get current speed
- Compute control values
- Update parameters
- Send adjustment value to throttle

3.5.2 Monitoring functions
- Global positioning system monitoring using GPS system
- Monitoring the user interface
- Monitoring the brake using brake sensor
- Monitoring the accelerator using accel sensor
- Engine monitoring using engine sensor
- Monitoring the speed using wheel revolution sensor

3.5.3 Control functions
- Comparing the current speed and desired speed, suitable control signal is generated either to increase or decrease the speed
- Update the parameters

3.5.4 Actuating functions
- Based on the control signal, throttle actuator is controlled to maintain the desired speed

4. APPROACH
4.1 Objectives
1. Design of fault tolerant model for CCS in
   i) TRS
   ii) ERMS models
2. Implementation of the proposed models
   i) Analysis
   ii) Simulation
3. Validating the performance of ERMS model over TRS with suitable performance metrics.

4.2 Assumptions
1. Constraints for redundancy are based on the number of parallelizable non-critical tasks and their utilizations.
2. Non critical periodic tasks are preemptable.
3. An existence of appropriate watchdog mechanism is assumed to be present that enables the detection of processor failures with a bounded latency.
4. Priority of non-critical task is assumed to be greater than the priority of the soft aperiodic task.
5. Hard aperiodic task is assumed to have the highest priority compared to critical tasks
6. A fixed time interval between the faults is assumed to be present in the system.
7. Execution time of each task is assumed to be worst case execution time(WCET) which includes all time overheads like context switches due to preemptions and communication costs between processors.
8. Each aperiodic task is assumed to a single event without any bursts.

4.3 Model schematic
Table 1: Task Set of CCS
P-periodic , Ap - Aperiodic tasks, Op – Periodic optional tasks

Sl. no	Tasks / nature (P/Ap/Op)	Processors	C_i	D_i	T_i
1.	Monitoring the Speed τ_{NC1} (P)	P1	3	15	30
2.	Monitoring acceleration τ_{NC2} (P)	P1	2	10	20
3.	Monitoring the CCS clutch τ_{NC3} (P)	P2	2	10	20
4.	Monitoring the brakes τ_{NC4} (P)	P2	3	15	30
5.	Monitoring proximity sensor τ_{NC5}(P)	P1	2	15	20
6.	Computing the control values τ_{C6} (P)	P1,P2	10	55	60
7.	Actuating the throttle valves τ_{C7} (P)	P1,P2	5	30	60
8.	Updating the parameters in τ_{NC8}(P)	P1	10	15	20
9.	Sensing the GPS data τ_{OP9} (Op)	P1|P2	2	-	20
10.	Computing the slope τ_{OP10}(Op)	P1|P2	10	-	60
11.	Brake Pedal Press τ_{HA11} (Ap)	P1,P2	1	10	10
12.	Change in speed due to uneven road conditions τ_{SA12} (Ap)	P1|P2	5	6	3

The task set of CCS with its task attributes is given in Table 1. The periodic tasks τ_1 to τ_5 are classified as non-critical sensing tasks that can be parallelizable. Tasks τ_6, τ_7 are classified as critical tasks which compute the control values and perform the actuating function, they are precedence constrained with sensing tasks. Tasks τ_9, τ_{10} are considered as optional tasks which improves the performance of the CCS by calculating the slope ahead using GPS mechanism. During the occurrence of faults, some of the optional tasks can be neglected to maintain the safe functionality of the system. Tasks τ_{11},τ_{12} are aperiodic tasks which are user or environment initiative. τ_{11} is classified as critical tasks since the brake pedal pressed by the driver has to be processed immediately to avoid the catastrophic effects.

Figure. 2 represents the fault tolerant model of CCS with a dual processor system and a master node for the given system.
Global clock - It provides synchronization between the components of the model
Health check - A self check logic circuit in each processor periodically sends the health status of the processor as an ALIVE signal to the global real time executive manager.

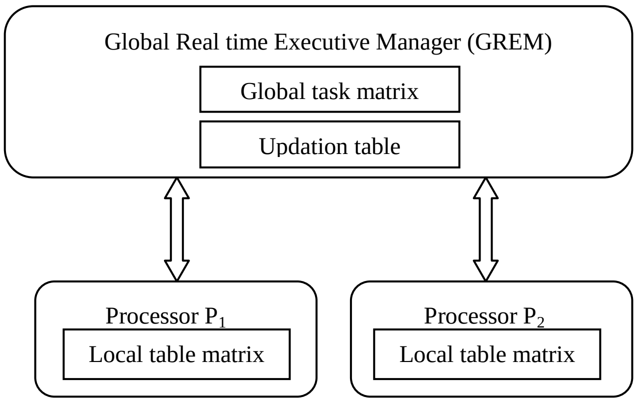


Fig 2: Schematic representation of the model

Global Real time executive manager (***GREM)*** - The ***GREM*** acts as a master node, maintains the global task table matrix, updation table matrix during runtime and monitors the health status of the processors. Absence of the **ALIVE** signal indicates the permanent failure of the processor under which a fault recovery algorithm reallocates the tasks of the failed units to the rest of the units by dynamic online reconfiguration.

Global table matrix - This table consists of all the tasks of CCS with its task attributes, nature and criticality of the task.

Updation table - This table indicates the remaining utilization of the processors and the level of criticality of each task that is executed in each processor. It is updated for every instance of time based on the information from each processor.

Local table matrix – Each processor is provided with a local table matrix which indicates the tasks that are to be executed by the processor.

4.3.1 Methodology for task allocation using TRS

In the traditional redundancy scheme, all the tasks in a task set are assigned and executed at the same instance in both the processors. The system continues to function with full functionality even during the failure of one processor. Algorithm **TaAl - TRS** represents task allocation in TRS in processors P_1, P_2.

Algorithm for Task allocation in TRS (**TaAl - TRS**)

```
Input: τ is a given task set of periodic tasks stored in GREM
Output: TRS schedule in normal mode
1: for i = 1 to n do
2:      schedule the task in both $P_1$ $P_2$
3: end
3: for each clock pulse do
4:      Trigger the transmission of ALIVE signal in $P_1$ and $P_2$
5: Update GREM
6: end
```

4.3.2 Methodology for task allocation using ERMS

In ERMS scheme, an innovative paradigm for load sharing using task level criticality is introduced, where critical tasks are replicated in both the processors and non-critical tasks are shared among the processors. The additional slack time which is made available by the ERMS is effectively utilized for scheduling the arriving aperiodic tasks and extra optional tasks. Algorithm **TaAl - ERMS** represents the task allocation in ERMS considering a dual processor system.

Algorithm for Task allocation (**TaAl - ERMS**)

```
Input: τ is a given task set of periodic tasks stored in GREM
Output: ERMS schedule in normal mode
1: for i = 1 to n do
2:      Check the criticality of task
3:       Set time($P_1$), time($P_2$) corresponding to the required
         workload of the processor
4:   if (Non-critical) then
5:      if (time($P_1$) > time($P_2$))
6:      Add task to Processor $P_1$
7:      if (time($P_1$) < time ($P_2$) then
8:      Add task to processor $P_2$
9:   if (Critical) then
10:      Add task to both $P_1$ and $P_2$
11:  for each clock pulse do
12:       Trigger the transmission of ALIVE signal in $P_1$ and $P_2$
13:  Update GREM
```

Under normal mode each processor continues to execute the tasks assigned to it until the arrival of aperiodic tasks. For each aperiodic task arrival ***GREM*** executes the *admission control* algorithm by checking the criticality of the arriving task and assigning it to the processors.

Algorithm for scheduling arriving aperiodic tasks (**Al-ADCT**)

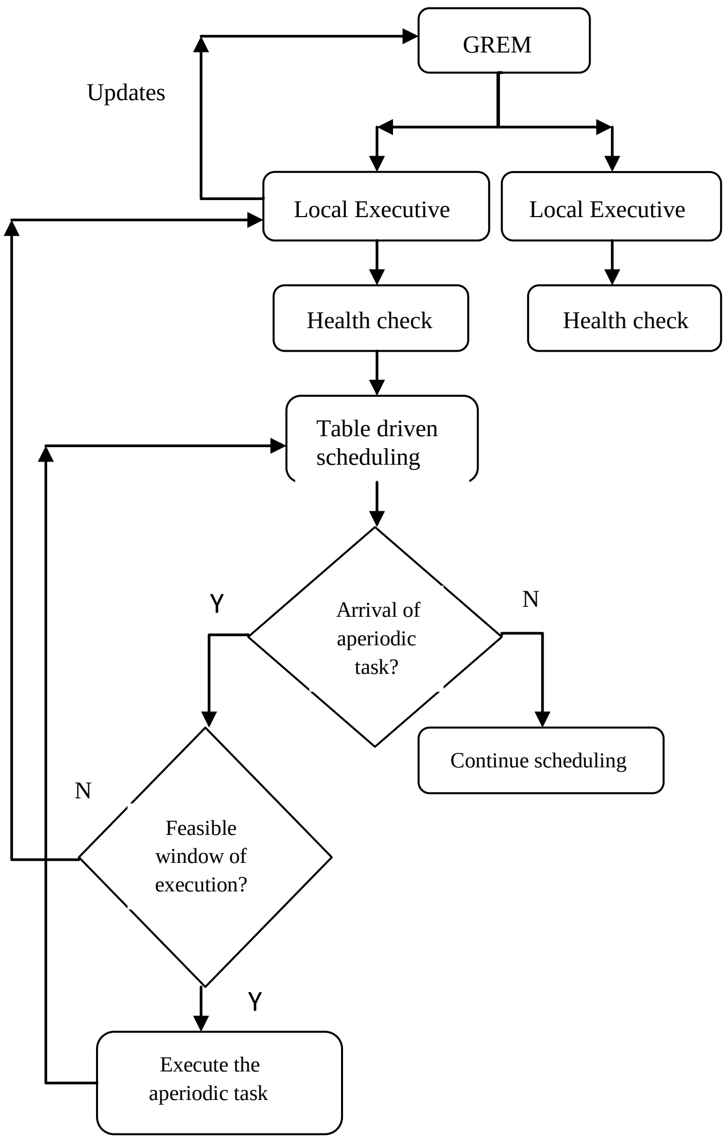
```
Input: Aperiodic task with known execution time and time
       period
Output: Selection of efficient processor for the execution of
        Aperiodic task
1: for arriving aperiodic task do
2:      Check the criticality of aperiodic task
3:  if (Critical) then
4:       schedule the task immediately in both the processors
5:  if (Non-critical) then
6:     if (time($P_1$) > time($P_2$)) then
7:       add the aperiodic task to the task queue of $P_1$
8:        schedule the aperiodic task during the available slack
          time of $P_1$
9:     if (time($P_1$) < time($P_2$)) then
10:       add the aperiodic task to the task queue of $P_2$
11:        schedule the aperiodic task during the available slack
          time of $P_2$
```

Figure 3 presents the flowchart of the proposed algorithm under normal mode. The local executive in both the processors contains the task table which it has to execute and follows a table driven scheduling. On the arrival of each soft aperiodic task, processor is checked for the feasible window of execution for scheduling the arriving task.

Under fault mode where one of the processors fails permanently, ***GREM*** reallocates all the non-critical tasks of the failed processor to the functioning processor.

Algorithm for fault mode

```
Input: An external or internal cause leading to permanent
failure of a processor
Output: Fault tolerant ERMS schedule
1: for i =1 to hyper period do
2:   if (Alive signal ($P_1$) is absent) then
3:        GREM reallocates the non-critical tasks of $P_1$ to $P_2$
          without violating the precedence constraints
4:   if (Alive signal ($P_2$) is absent) then
5:        GREM reallocates the non-critical tasks of $P_2$ to $P_1$
          without violating the precedence constraints
```



5. IMPLEMENTATION
The fault tolerant models proposed in section 4 are analyzed and simulated using suitable tool box.

5.1 Analysis
Scheduling Analysis of the models of the two fault tolerant schemes and their corresponding results are shown in Figure 4. Figure 4(a),(b) represents the normal mode scheduling of ERMS and TRS with the given task allocation algorithms. Figure 4(c),(d) represents the scheduling of hard aperiodic task arrival at 5th time unit in ERMS and TRS. Figure 4(e),(f) represents the error recovery scheduling when fault occurs in processor P_1 at 5th time unit for ERMS and TRS respectively.

■ Critical ▨ Non-critical 🟪 Optional tasks

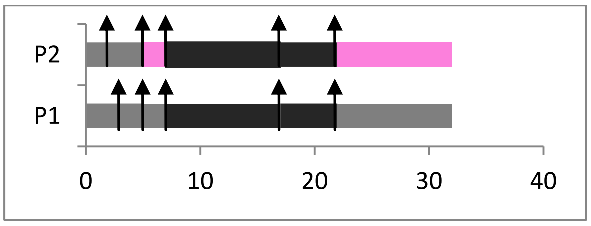


Fig 4(a): ERMS scheduling under normal mode

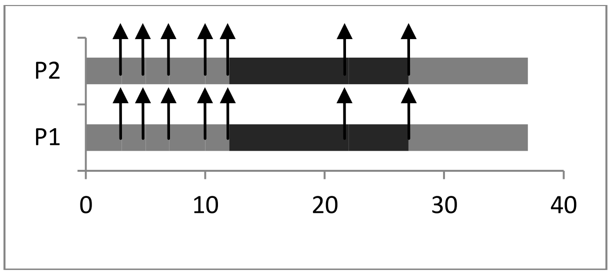


Fig 4(b): TRS scheduling under normal mode

🟩 Hard Aperiodic task

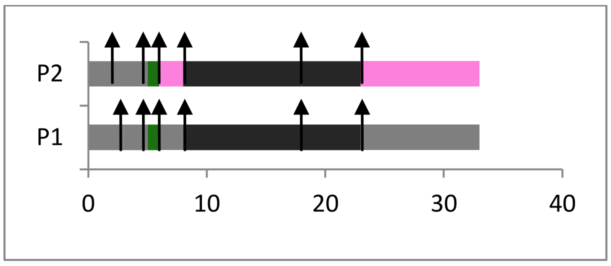


Fig 4(c): ERMS scheduling with hard aperiodic task arrival at 5th unit

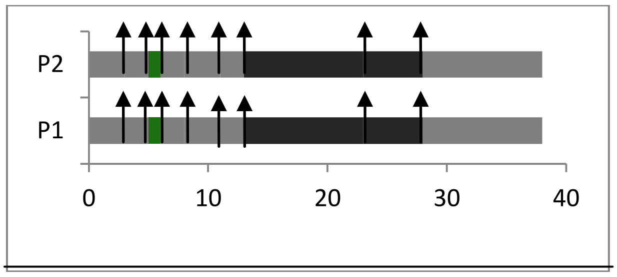


Fig 4(d): TRS scheduling with hard aperiodic task arrival at 5th unit

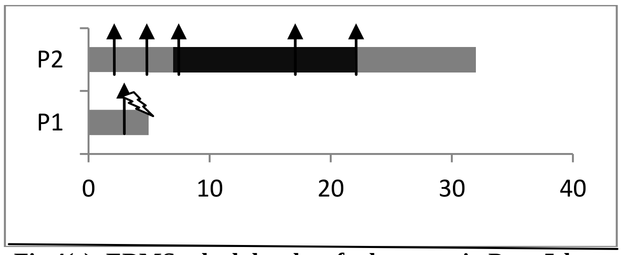


Fig 4(e): ERMS schedule when fault occurs in P_1 at 5th unit

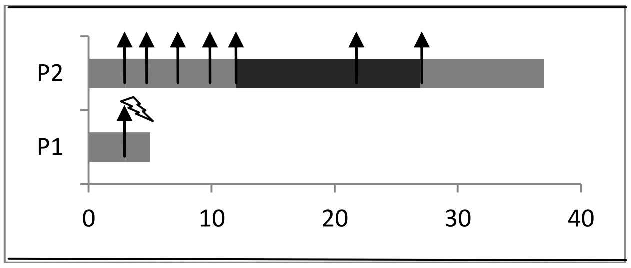


Fig 4(e): TRS schedule when fault occurs in P_1 at 5th unit

5.2 Simulation
The simulation of the proposed fault tolerant algorithm is carried out in Matlab with the help of Time Optimisation Resource and SCHEduling (TORSCHE) toolbox. The periodic task set in Table 1 is input to the global task table in

the GREM. The work load is assumed to be 80% and the two processors have been assigned a set of tasks based on criticality (Section 4) table driven scheduling has been implemented in both the processors. Graphical user interface(GUI) have been designed to project the scheduling results during various working conditions of the system.
Figure 5 represents the simulation results of the ERMS in Matlab where all the tasks of CCS are grouped into a single task set considering the precedence constraints. Figure 5(a) represents ERMS scheduling under normal mode where critical tasks are duplicated on both the processors and non-critical tasks are shared among the processors using load balancing techniques. Each sensing task is assumed to be storing its results in a particular memory location specified in wrapping code associated with each task. The control task τ_6 which is dependent on all the sensing tasks is assumed to fetch the results through inter process communication. Figure 5(b) represents TRS scheduling strategy where all the tasks τ_1 to τ_8 are duplicated in both the processors.

■ Critical ▩ Non-critical 🟪 Optional tasks

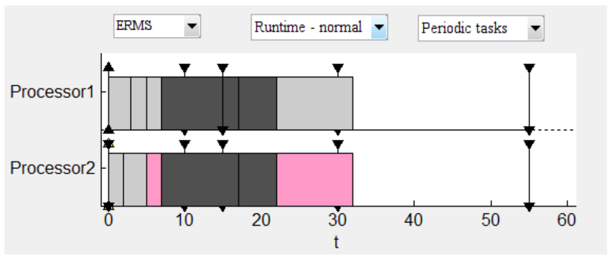

ERMS | Runtime - normal | Periodic tasks

Fig 5(a): ERMS under Normal mode with periodic tasks

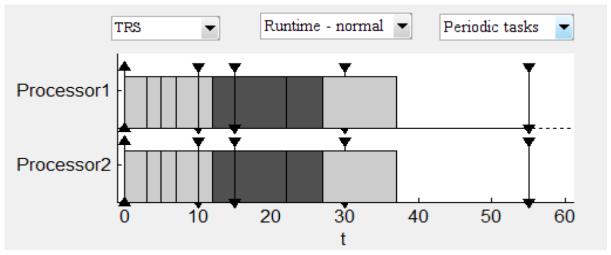

TRS | Runtime - normal | Periodic tasks

Fig 5(b) : TRS under Normal mode with periodic tasks

🟩 Hard Aperiodic task 🟦 Soft Aperiodic task

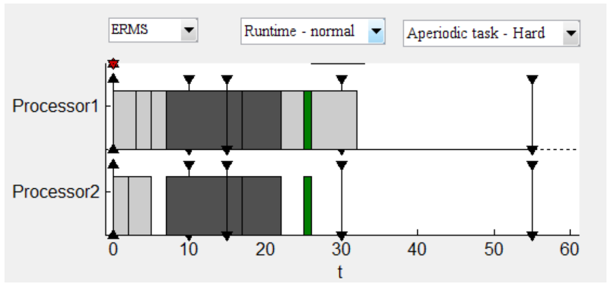

ERMS | Runtime - normal | Aperiodic task - Hard

Fig 5(c): ERMS under Normal mode with hard aperiodic task at 25th unit

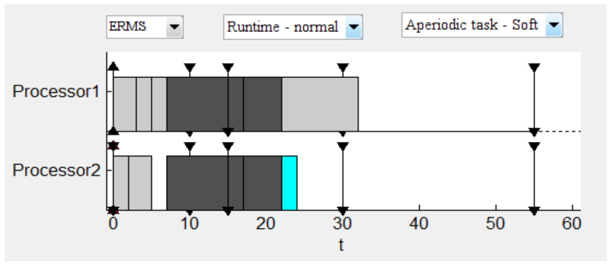

ERMS | Runtime - normal | Aperiodic task - Soft

Fig 5(d): ERMS under normal mode with soft aperiodic task at 25th unit

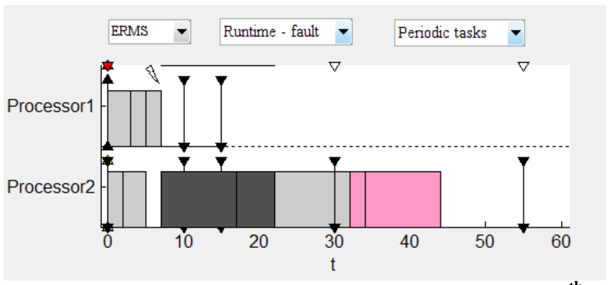

ERMS | Runtime - fault | Periodic tasks

Fig 5(e): ERMS under Fault mode with P_1 failure at 7th unit

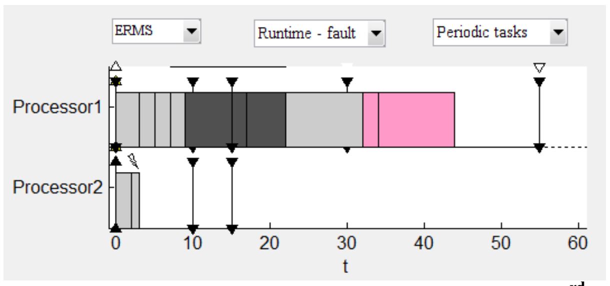

ERMS | Runtime - fault | Periodic tasks

Fig 5(f): ERMS under Fault mode with P_2 failure at 3rd unit

Figure 5(c) represents ERMS scheduling with an hard aperiodic tasks τ_{HA11} arriving at 20th time unit with an execution time of 1unit. The arriving aperiodic task is scheduled on processor P_1, P_2. Figure 5(d) represents ERMS scheduling with a soft aperiodic task τ_{SA12} arriving at 20th time unit with an execution time of 5 units. It is scheduled on processor P_2 in the available slack time at 32nd time unit. Figure 5(e) represents ERMS scheduling algorithm in fault mode where processor P_1 is considered to have failed after the completion of execution of task τ_{NC5} at 7th time unit. ***GREM*** reallocates the non-critical tasks τ_{NC1}, τ_{NC2}, τ_{NC5}, τ_{NC6} of processor P_1 to P_2.
Figure 5(f) represents ERMS scheduling with an occurrence of failure on processor P_2 in between the execution of task τ_{NC4} at 3rd time unit. ***GREM*** reallocated the remaining non-critical tasks to Processor P_1 and τ_{NC4} is resumed back in P_2 followed by the critical tasks. In the next hyper period ***GREM*** reschedules all the non-critical task on processor P_1.

The simulation results reveal the effectiveness of the proposed ERMS scheme in comparison with the TRS scheme. Under normal mode it increases the speed of the execution by efficiently utilizing the available processors. As a result of this, *the execution time of the entire process of CCS reduces to 32time units in comparison with 37time units of TRS algorithm*. It also provide an *extra slack time of approximately 13% in comparison with TRS algorithm. In fault mode the system operates with minimal safe functionality by making sure that all the critical periodic and aperiodic tasks meet their stringent deadlines.*

6. EVALUATION

6.1 Process speedup S_p

Figure 6 indicates the execution time of the application by ERMS and TRS given as 32 and 37 time units respectively. It shows that the ERMS speeds up the process by reducing the total execution time of the process by 13% over the TRS.

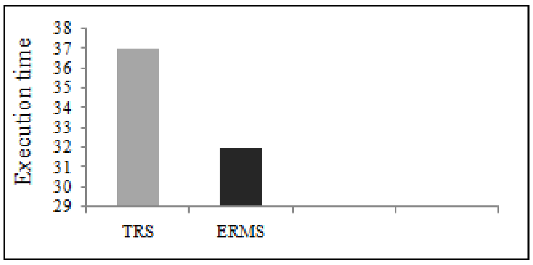


Fig 6: Process Speedup of TRS and ERMS

6.2 Effective Utilization U_e

Figure 7 indicates the normalized utilization of TRS and ERMS as 84.3% and 54.3% respectively. The ERMS provides an additional 33% computing time for execution of extra *optional tasks* as compared to TRS.

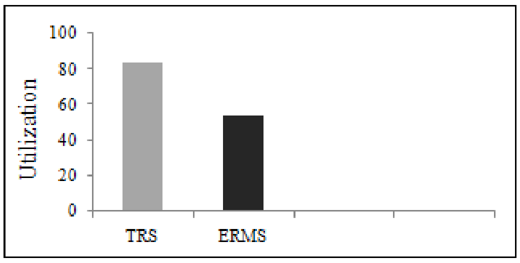


Fig 7: Effective Utilization of TRS and ERMS

6.3 Average response time of aperiodic tasks R_a

Figure 8 indicates the average response time of the system to arrivals of soft aperiodic tasks for TRS and ERMS scheme. It shows how ERMS effectively reduces the average response time of soft aperiodic tasks as compared to TRS.

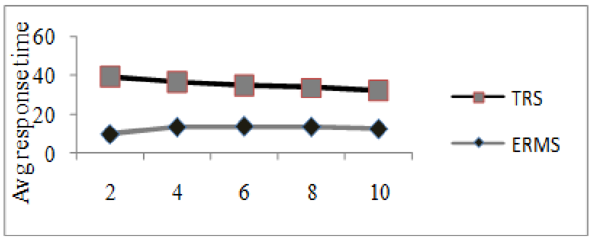


Fig 8 : Average response time of aperiodic tasks

6.4 Guarantee ratio G_e

Figure 9 indicates the guarantee ratio of ERMS with varying workloads. At a minimal workload of 60%, ratio is approximately 1 in both normal and fault mode. As the work load increases guarantee ratio decreases with minimal safe functionality being maintained. At 80% workload, the ratio decreases by 50% in normal mode and 70% in fault mode.

6.5 Deadline miss ratio D_r

Figure 10 indicates the deadline miss ratio of ERMS with varying workloads. At a workload of 60%, there is no deadline miss occurring in normal mode. As the work load, there is an increase in deadline miss ratio by 50% in normal mode and 66% in fault mode for a workload of 80%.

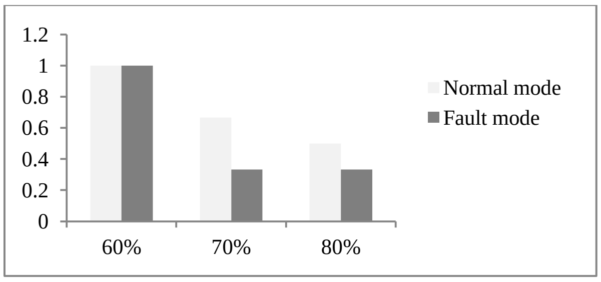


Fig 9 :Guarantee ratio

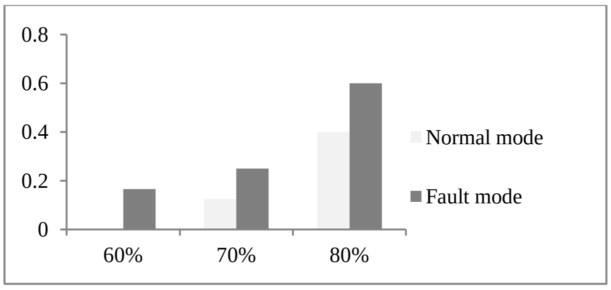


Fig 10: Dead line miss ratio

7. CONCLUSION

In this paper, a fault tolerant multiprocessor scheduling of cruise control system with effective resource management has been proposed. The comparison of performance metrics of ERMS over a traditional redundancy scheme demonstrates the improvement in the performance which provides an additional 33% computing time resources for the execution of extra *optional tasks* as compared to TRS. The ERMS speeds up the process by reducing the total execution time of the process by 13% over the TRS under runtime normal mode. The response time of the soft aperiodic tasks is reduced by implementing the dynamic planning based approach and efficiently utilizing the extra slack margin available. The results conclude the gains that can be obtained in terms of high performance throughput and process speed up if this model is extended to an m-processor redundancy model specially in the field of avionics where the advantages can be highly appreciable in terms of fuel/weight ratio. Further this algorithm can be extended to more complex applications like avionics, missile launching, etc. to improve the system performance.

8. REFERENCES

[1] J.W. Layland and C. L. Liu, "Scheduling algorithms for multiprogramming in hard real-time environment", *Journal of the* ACM 20 (1973), no. 1, 46-61.

[2] C.L. Liu, "Scheduling algorithms for multiprocessors in a hard real-time environment". JPL Space Programs Summary, vol. 37-60, pp. 28-31, 1969.

[3] C.A. Phillips, C. Stein, E. Torng, J. Wein, "Optimal time-critical scheduling via resource augmentation". In Proceedings of the ACM Symposium on theory of Computing 1997.

[4] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, *Annals of Math Studies*, numbers 34, pages 43-98. Princeton Univ. Press, 1956.

[5] E.F. Moore and C.E. Shannon. Reliable circuits using less reliable relays. *J. Franklin Institute*, 262:191-208 and 281-297, Sept/Oc. 1956.

[6] W.H. Pierce. *Failure-Tolerant Computer Design.* Academic Press, 1965.

[7] Avizienis A., "Design of Fault-Tolerant Computers" Fall Joint Computer Conference 1967 [Aviz67].

[8] D. Mosse, R. Melhem, and S. Ghosh, "Analysis of fault tolerant multiprocessor scheduling algorithm" Proc. IEEE Real Time Systems Symp, pp.129-139, Dec.1981

[9] C. M. Krishna and K. G. Shin, "On scheduling tasks with a quick recovery from failure" *Proc. Fault-tolerant Comput. Symp,* pp. 234-239, 1985.

[10] Y. Oh and S. H. Son, "Enhancing fault-tolerance in rate monotonic scheduling," *Real-Time Systems*, vol. 7, no.3, 1994.

[11] Y. Oh and S. H. Son, "Scheduling real-time tasks for dependability," *Journal of Operational Research Society*, vol. 48, pp. 629-639, 1997.

[12] Manimaran G, "Fault tolerant dynamic schedule for multiprocessor real time systems and its analysis" Parallel and Distributed Systems, IEEE Transactions (vol 11) 1998.

[13] Mahmud Pathan, "Three Aspects of Real-Time Multiprocessor Scheduling: Timeliness, Fault Tolerance, Mixed Criticality" *Göteborg, Sweden, 2012* ISBN: 978-91-7385-754-3.

[14] Radhamani Pillay, Sasikumar Punnekkat, Senthil Kumar Chandran, "An improved redundancy scheme for optimal utilization of onboard Computers", IEEE INDICON 2009, India

[15] Radhamani Pillay, Senthil Kumar Chandran, and Sasikumar Punnekkat, "Optimizing resources in real-time scheduling for fault tolerant processors", IEEE, International Conference on Parallel, Distributed and Grid Computing (PDGC-2010), Solan India; October2010.

[16] Senthil Kumar Chandran, Radhamani Pillay, Radu Dobrin, and Sasikumar Punnekkat, "Efficient scheduling with adaptive fault tolerance in heterogeneous multiprocessor systems", International Conference on Computer and Electrical Engineering (ICCEE) Chengdu, China; Nov 2010. China.

[17] Technical Report: "Adaptive Cruise Controllers – A Literature Review", Stefan Björnander, Mälardalen University, Sweden 2008.

[18] N. Audsley, A. Burns, "Real time scheduling", Department of Computer Science,University of York, UK.

[19] Krithi Ramamritham, member, IEEE, and John A. Stankovic, fellow, IEEE "Scheduling Algorithms and Operating Systems Support for Real-Time Systems"

[20] Sherin Abraham, Radhamani Pillay, "Fault Tolerance in Safety Critical Applications*"* 2nd Intl. Conference on Advanced Computing and Communication technologies

for High Performance Applications, December 2011, Cochin, India.

[21] John C.Knight, "Safety Critical Systems: Challenges and Directions", Department of computer Science, University of Virginia.

[22] Gurulingesh R, Neera Sharma, Krithi Ramamritham and Sachitanand Malewar, "Efficient Real-Time Support for Automotive Applications: A Case Study", Indian Institute of Technology Bombay.

[23] Robert I. Davis, Alan Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems", Real-Time Systems Research Group, Department of Computer Science, University of York, York, UK.

[24] G.Manimaran, C.Siva Ram Murthy, " A New Study for Fault-tolerant Real-time Dynamic Scheduling Algorithms", Department of Computer Science and Engineering, Indian Institute of Technology, Madras.

[25] Brinkley Sprunt, "Aperiodic Task Scheduling for Real-Time Systems", Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University.

[26] Chenyang Lu, John A. Stankovic, Tarek F. Abdelzaher, "Performance Specifications and Metrics for Adaptive Real-Time Systems", Department of Computer Science, University of Virginia, Charlottesville.

[27] Anthony Spiteri Staines, "Modeling and Analysis of Real Time Control Systems: A Cruise Control System Case Study", University of Malta, Malta.