Resource Allocation Optimization through Task Based Scheduling Algorithms in Distributed Real Time Embedded Systems

A THESIS
SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR DEGREE OF

BACHELOR OF TECHNOLOGY

IN THE DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

by

Swagat Mishra
107CS010

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India

May 2011

Supervisor: Prof. P. M. Khilar

CERTIFICATE

This is to certify that the work in the thesis entitled **'Resource Allocation Optimization through Task Based Scheduling Algorithms in Distributed Real Time Embedded Systems'** by **Swagat Mishra**, **Roll No: 107CS010** is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science Engineering** at **National Institute of Technology, Rourkela.**
To the best of my knowledge the matter embodied in the thesis has not been submitted to any other university/institute for the award of any degree or diploma.

Prof. Pabitra Mohan Khilar
Department of Computer Science and Engineering,
National Institute of Technology,
Rourkela-769008

ACKNOWLEDGEMENT

Words will do no justice to the constant support and encouragement extended by Prof. P. M. Khilar who guided this thesis. If the work has seen light today, it is only because of his motivation during times of hardship.

Learning is a gradual process of assimilation of learned concepts and cultivating it into the knowledge base. It was an honour to learn from illustrious individuals like Prof. A. K. Turuk,Prof. M.N Sahoo and Prof. S. K. Rath. Whatever little I could learn from them surely helped me smoothen the learning curve, broaden horizons and sustain interest and passion in my work.

I also express my indebtness to my friends who playfully taught me several lessons of a life, something that is a treasure trove for me. Their indirect help in this thesis is worthy of mention.

I would also like to acknowledge the help of Prof. P.K. Sa without whom this thesis would not look as good as it does today.

Finally, a special thaks to the Almighty for seeing this work through and renewing satisfaction and confidence in me.

Swagat Mishra

Contents

Certificate **i**

Acknowledgement **ii**

List of Figures **iv**

List of Algorithms **v**

1 INTRODUCTION **1**

1.1 Motivation 2
1.2 Overview 2
1.3 Thesis Objective 3
1.4 Assumptions 3
1.5 Thesis Organisation 4
1.6 Conclusion 4

2 LITERATURE REVIEW **5**

2.1 Scheduling Algorithms 6
2.1.1 Graph Folding 6
2.1.2 Algorithm 6
2.2 Fault Detection 6
2.3 Fault Recovery 7
2.3.1 Uncoordinated Checkpointing 7
2.3.2 Coordinated Checkpointing 8
2.4 Temperature Control 8
2.5 Conclusion 9

3 System and Fault Model **10**

3.1 System Model 11
3.2 Fault Model 12
3.3 Conclusion 12

4 Algorithm **13**

4.1 Algorithm Description 17
4.1.1 Types of Nodes 17
4.2 Heartbeat and checkpointing mechanism 18

4.3 Algorithm Analysis 18
Before setup of pipeline 19
After setup of pipeline 19
4.4 Conclusion 19

5 Simulation **20**
5.1 Implementation 21
5.2 Simulation Results 22
5.3 Implementation Analysis 25
5.4 conclusion 25

6 Conclusion **26**
6.1 Conclusion 27

List of Figures

3.1 System Model 11

4.1 Processor Interaction Diagram 17

5.1 Fault Injection Simulation 23
5.2 Messages passed with pipeline not set 23
5.3 Optimized Algorithm vs Old Algorithm 24

List of Algorithms

1 Working of Monitor Node 14
2 Working of Processor Node 16

Abstract

Distributed embedded system is a type of distributed system, which consists of a large number of nodes, each node having lower computational power when compared to a node of a regular distributed system (like a cluster). A real time system is the one where every task has an associated dead line and the system works with a continuous stream of data supplied in real time.Such systems find wide applications in various fields such as automoble industry as fly-by-wire,brake-by-wire and steer-by-wire systems. Scheduling and efficient allocation of resources is extremely important in such systems because a distributed embedded real time system must deliver its output within a certain time frame, failing which the output becomes useless.In this paper, we have taken up processing unit number as a resource and have optimized the allocation of it to the various tasks.We use techniques such as model-based redundancy,heartbeat monitoring and checkpointing for fault detection and failure recovery.Our fault tolerance framework uses an existing list-based scheduling algorithm for task scheduling.This helps in diagnosis and shutting down of faulty actuators before the system becomes unsafe. The framework is designed and tested using a new simulation model consisting of virtual nodes working on a message passing system.

Chapter 1

INTRODUCTION

1.1 Motivation

Distributed embedded systems (DES) is a vastly important topic and represents a revolution in the field of computing and information technology(IT) [1, 2].DES are networks of embedded computers whose functional components are nearly indivisible to end users. they have the potential to alter radically the way in which people interact with their environment by linking a range of devices and sensors that will allow information to be collected, shared, and processed in unprecedented ways. Widespread use of DES is fast spreading through the computing area and is indicative of the long term trend of moving away from centralized, high cost, low volume products towards distributed, low-cost, high-volume products. It could easily dwarf previous revolutions in IT for two main reasons- Firstly, Moores law is primarily driving device miniaturization and reduced power consumption (instead of increased speed). Secondly, the industry is fast becoming better equipped with development tool chains (software included) and pervasive standards-based communications protocols. As a consequence, embedded computing and communications technology can quite literally be everywhere and extend into all aspects of lifean invisible component of almost everything in everyones surroundings. Distributed embedded real time systems handle jobs that have deadlines (hard or soft) associated with them. Lately such systems have found application in a number of sectors. They will be the informational power grids for the 21st century. The major industry sectors where DES is likely to be used are automotive, avionics/aerospace, industrial automation (and robotics); telecommunications; consumer electronics and intelligent homes; health and medical systems. In this thesis the applications of Distributed embedded real time systems in the field of automotive and aerospace sectors is of most importance. In the automotive sector, DES is used for transmission, airbags, antilock braking, cruise control, audio systems, windows, doors, mirror adjustment etc. The DES units may be used in the form of sensors, actuators or other control devices. An aerospace model may consist of such activities like launch of a rocket or an aeroplane. These are real time applications of DES and can incorporate the algorithms and frame work proposed in this paper for efficient and fault tolerant scheduling.

1.2 Overview

In the automobile industry distributed embedded systems have very high safety constraints to avoid accidents due to faults like unintended steering of the vehicle.This presents several challenges to the developement of an efficient error diagnosis mechanism.Total Processor cost is another important consideration which has to minimized by minimizing the number of processors used in the system,without compromising the safety of the system.The temperature of individual nodes requires careful monitoring as well. Uncontrolled increase in temperature can lead to degraded performance and decreased reliablity of the system [3].It can also damage the nodes of the DES resulting in damage to the Steer-by-Wire system itself [4].To this end,model based redundancy [5] is used which uses the discrepancies between the results of the ac-

tuators and a mathematical model to detect faults.However,using redundancy alone as a fault detection technique leads to wastage of processing units as the same task is being done by a number of processing units.To this end, heartbeat monitoring is being used as an extra layer of fault detection measure.This will enable us to reduce the redundancy that is employed in the system.Failure detection is done as usual by checkpointing.

1.3 Thesis Objective

The various objectives that this thesis accomplishes are

- Given a system model as shown in figure 3.1 and task graphs as in [6],we wish to design a general framework based on a monitor module.The framework reduces the level of redundancy needed in the system, by detecting and removing faulty nodes in the system.
- We also present a dynamic scheduling algorithm based on [6],that allows rescheduling of tasks with changes in the release time,execution time and deadline constraints.The types of faults dealt with are crash faults and transient faults in the system.
- The proposed framework is implemented and analysed using message passing interface(OpenMPI) libraries.Analysis is done on the basis of number of messages passed between nodes and the response of the system to faults.

1.4 Assumptions

To achieve the above objectives,we make the following assumptions

- The distributed embedded system's nodes are connected by a reliable communication network.
- The processors are susceptible to crash and transient faults whcih can cause them to be permanently or temporarily decapacitated resulting in errors.However a faulty processor does not effect the remaining healthy ones.
- A stable storage is present to store the global state during checkpointing.
- During the execution of a task,if at most k faults may occur to the processors executing the task during the corresponding iteration then the task is executed by at least k processors.This assumption limits the number of faults to k during a single iteration only.Another k faults can occur during subsequent iterations as well.
- The various tasks,including control and diagnostic tasks, can be pre-empted and resumed from a suspended state, provided their execution deadline is not missed.

1.5 Thesis Organisation

This thesis has been divided into 6 chapters.Each chapter explains a different aspect of this thesis.
Chapter 1 introduces the topic of fault tolerance in distributed embedded systems.It explains the applications of distributed embedded systems and the motivation behind this thesis.It also provides the objectives of the thesis and explains what the thesis strives to achieve.
Chapter 2 deals with the literature and related works on which this thesis is based on.It provides the various fault tolerance mechanisms and scheduling algorithms that are used in the design of the fault tolerance framework.It covers scheduling,fault detection,fault recovery and temperature control of the distributed embedded system.
Chapter 3 describes the system and fault models for the system.
Chapter 4 provides the pseudo code of the entire algorithm.It describes and analysis the algorithm and gives in detail the checkpoint and heartbeat mechanism used in the algorithm.
Chapter 5 describes the simulation methodology used to validate the system.It provides the simulation results and analyses the results.
Chapter 6 concludes the thesis and provides the conclusion from the simulation and analysis results.

1.6 Conclusion

Distributed embedded systems have inportant applications in automobile and other sectors which makes it critical to make such systems fault tolerant.Besides the automobile sector, distributed embedded systems also find applications in a large number of fields such as medical etc.In this chapter,we have examined the various objectives of this work and the assumptions required to achieve these objectives.This chapter also details the organisation of the entire thesis.

Chapter 2

LITERATURE REVIEW

2.1 Scheduling Algorithms

The scheduling algorithm used in the thesis is adapted from [6].It may be divided into 2 parts

2.1.1 Graph Folding

Graph folding is used during scheduling of tasks on the execution pipeline. It is preferred over graph unrolling because it does not introduce any jitter.To unroll a graph, the tasks are executed such that tasks belonging to different iterations are executed together in a schedule. This involves controlled unrolling of the graph to expose several iterations of the graph.The tasks are scheduled in a frame of constant size.Tasks that cross the frame boundary are considered as separate tasks and folded to fit in the frame boundary.For a detailed discussion of graph folding,see [7]

2.1.2 Algorithm

The algorithm as given is [6] is a static,list scheduling algorithm.It schedules the tasks on a first in first out basis.Before assigning a task to a processor, it checks if the various constraints related to the task are satisfied and the preceding tasks are completed.If these conditions are satisfied,the task is assigned to a processor.The execution occurs in time frames.The tasks are fit into the time frames and the tasks that cross the time frame boundary are folded to fit into the time frame.We make changes to this algorithm and make it dynamic by allowing the processing nodes to check the execution time of the tasks.If the actual execution time varies from the time given in the pipeline by a certain threshold, the processing unit may notify the monitor node to invalidate the pipeline and recalculate the schedule.

2.2 Fault Detection

For fault detection, we are using hearbeat monitoring.Our heartbeat monitoring system is partially based on the "lazy" heartbeat approach as given in [8].
The type of failure detectors used in our fault detection algorithm is an example of a hybrid "push and pull" mechanism.In this mechanism,a fault detector node queiries another node to check if the node is "alive" or not.From the acknowledgement to this heartbeat message, the fault detector can trust the node to be fault free for a certain interval of time Δ_{trust}.After this interval, the detector sends another heartbeat signal and the process goes on.If a timeout occurs to the receipt of a heartbeat signal by the detector, then it puts the node in the list of nodes suspected of being faulty.Faults can be confirmed by the use of successive heartbeat signals.Our approach has higher message overhead than a failure detection system based only on the "push" mechanism.However,it has the significant advantage of being able to detect faults in the processor as well as monitor nodes."Lazy" monitoring approach tries to send as few heartbeats as possible to reduce the overall message overhead.This is done by treating

the application messages being sent to the processor nodes are treated as heartbeat signals.This can be achieved by keeping the inter-heartbeat interval Δ_i large enough so that any 2 application messages fall in the inter-heartbeat interval.This is useful in the case of distributed embedded systems in which a only a few faults have occured and there is sufficient redundancy in the system to mask any faults.As given in [8], the heartbeat mechanism is susceptible to errors due to message delay,message loss and processing delay. All of these must be taken into account while determining the inter-heartbeat receipt times.For our purposes, we have modified the heartbeat sample duration equation to look like:
$s = \Delta_i + l * \Delta_i - f * \Delta_f$
where Δ_i is the initial heartbeat interval,
$l * \Delta_i$ is the message loss rate with l being the successive messages lost,
f is the number of nodes that have been found to be faulty and hence have resulted in a decrease in redundancy for the task,
and Δ_f is the increase in sampling rate that must be effected to counter the loss in redundancy. This basically means that for a message loss which is the charecteristic of the network, we take delayed samples to allow heartbeats to reach the monitoring nodes after incurring message losses.However, samples are taken at a quicker rate after a fault occurs in the processing nodes.This can be done by switching from using application messages as heartbeats to using actual heartbeat signals sent by the monitor nodes.

2.3 Fault Recovery

Fault recovery is done via checkpoint based protocols,details of which are availiable in [8].we present a brief overview of the various protocols we use in our fault tolerance framework.The basic principle behind failure recovery is storage of a globally consistent system state.It is defined as one in which if the sender of a message reflects sending of a message ,then this must be reflected in the receiver and vice-versa [8].An inconsistent state is generally charecterized by orphan messages,which after roll back seem to have been received by a node but do not have a sender.To avoid this, after rollback if a sender is rolledback to a state where it has not sent a message,then the corresponding receiver must also be rolledback to a state where it has not received the message.For our purpose,2 types of checkpointing protocols are of interest

2.3.1 Uncoordinated Checkpointing

In uncoordinated checkpointing, the nodes have the freedom to take checkpoints without consulting other nodes.No central node is present to supervise the checkpointing of various nodes.It has the advantage of having minimum communication overhead, as the nodes do not have to pass messages to determine when to take checkpoints.It is also very simple to implement.However, this method has 2 chief disadvantages.One is that ,without co-ordination, the nodes may take a checkpoint which is not part of a global consistent state and hence is an useless checkpoint.Secondly, this type of

checkpointing gives rise to the domino effect [9].When a node rollsback to a previous state, it may make the state of another node invalid,which must rollback to a previous state to maintain consistency.This inturn may trigger rollbacks in other nodes,the entire process leading to loss of a large amount of computation.This is called as domino effect and is a major problem of uncoordinated checkpointing.

2.3.2 Coordinated Checkpointing

In this protocol, the nodes take checkpoints in collaboration with each other so as to ensure a global consistent system state.Usually, a monitor node is present which coordinates the nodes to take consistent checkpoints.Our checkpointing protocol is partially based on the cooridinated checkpointing protocol approach as suggested by [10] .In this protocol, a central coordinator takes a checkpoint and sends messages to all nodes to take respective checkpoints.On receipt of this message, the nodes block all communications, take a checkpoint and send an acknowledgement to the coordinator which finally commits all the checkpoints.In our case, interprocessor communication is not of much importance as the jobs are being executed parellely and independently.Hence,our approach consists of a monitoring node instructing other processing nodes to take checkpoints.The checkpoint data is received as acknowledgement by the monitor,which validates the checkpoints.
Coordinated checkpointing has several advantages.It does not suffer from the domino effect.It also has less storage overhead than uncoordinated checkpointing.However it has high message overhead due to the communcations between coordinator and other nodes prior to taking a checkpoint.

2.4 Temperature Control

Overheating of processing units is an inevitable problem.The various problems due to overheating of processing units have been discussed earlier.The different processor utilization for the processing units result in different thermal profiles for each node.overheating of nodes can result in physical damage to the processing units which would increase the cost of the system.2 strategies are being used for heat control.The first is simple shutdown of overheated nodes.once the temperature of a node crosses the safety threshold,it is shutdown by the monitoring system and is replaced by a spare processer.The task schedule migration occurs after the end of the current iteration.
Heat control is also done by the use of schedule swapping [11].This technique has been shown to reduce the average temperature of the distributed embedded system by 11°C.In this method,The processors are sorted in order of their temperature. Then the task schedules of the hottest processer and the coldest processer are exchanged.This processer is done for all pair of hot and cold processors, to ensure uniform rise in temperature for all processing units.
Either one or both of these methods may be used to control temperature in the system.

2.5 Conclusion

This chapter details our literature survey into the various fields required for our thesis.We examine in details concepts such as fault recovery,fault detection,temperature control and scheduling algorithms.This helps us adapt the various fault tolerance strategies for our framework.This literature survey also gives the various modifications that we make to established fault tolerance techniques, so that they can be used in our framework.Thus, this chapter shows how different fault tolerance techniques fit together flawlessly in our framework.

Chapter 3

System and Fault Model

3.1 System Model

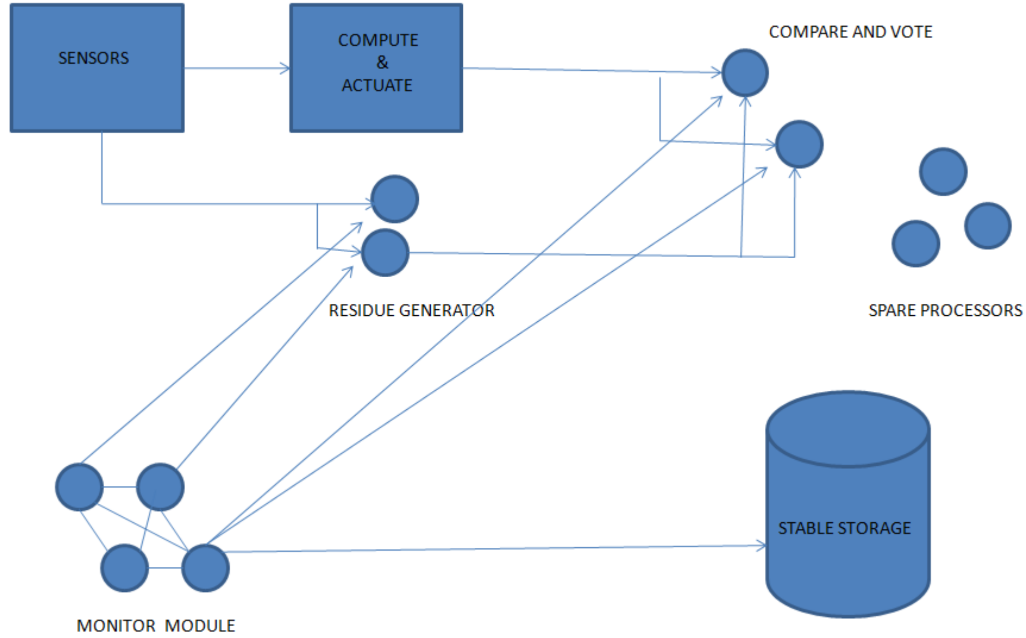

Figure 3.1: System Model

A distributed approach to fault diagnosis in actuator control is given in [6].However,we use a completely message passing based approach to diagnose faults.The system model is as shown in figure 3.1.

The system comprises of sensors,acuators,residue generators,voters,stable storage device and a monitor module.Redundancy is used by using a number of redundant and possibly diverse sensors to collect data,which is passed onto the "compute and actuate" block .In the "compute and actuate" block,results are computed from the sensor data and executed by the actuator.Sensor data is also passed into a residue generator which computes model-based results.These results are compared with the actuator results and put to vote.If a deviation is found from the model based results in the actuator results, a fault is concluded to have occured and the faulty actuator is shutdown.Analytical redundancy is implemented for faults occuring per iteration.If k faults occur in an iteration of the graph, we have k+1 processors executing the task to enable succesful detection.

The monitor module nodes are fully connected to each other and also to the processing units.They are responsible for set up of the schedule pipeline,taking checkpoints and heartbeat monitoring of the nodes.Faulty nodes in the monitor system are detected by the processing units.They expect heartbeat signals from the monitor nodes at certain intervals of time.If heartbeat signals are not sent at such times,the processing units assume the monitor node to be faulty and request other monitor nodes to replace it. Some spare processors are present in a "turned off" state to provide replacement nodes

for overheated or faulty nodes.They are connected to the communication network but have no functions till they are turned on by the monitor.Overheated nodes also form a part of the spare processors set till they cool down and are ready to work again.Faulty nodes are replaced by task migration during slack time.
The stable storage provided is an abstraction for a magnetic disk or a set of processors.It is required to store checkpoint data reliably and consistently.It should remain fault free even when the rest of the system goes down due to faults,so that the data in the stable storage may be used to resume execution.
The entire system is connected by a communication network such as shared bus.Messages are exchanged between processors by a reliable protocol such as a TDMA protocol.

3.2 Fault Model

The system is designed to handle 2 kinds of faults

- Crash Faults These fault occur due to a permanent failure in a processing unit of the system.The affected processor stops sending and receiving messages and can no longer take part in processing.Such faults normally occur due to physical damage to the processing unit such as in case of overheating.In our framework, they are taken care of by the monitor module,which replaces the crashed processors.Since the processors are fail-silent,any timeout in receiving heartbeat signals from the processors can be used to suspect a fault.

- Transient Faults These faults are random in occurance and disappearance.Unlike crash faults,they are temporary and may disappear at any time.These faults generally result in calculation errors in the task being executed in the processor when the fault occurs.These faults may occur due to random factors and may be corrected by handling the incorrect results they produce.This is done by the use of masking resundancy so that when one of the processors gives an incorrect result due to a transient fault,the results from fault-free nodes may be used to detect the fault node by polling and aslo ensure that the task execution results are fault free.

3.3 Conclusion

This chapter explains the system model and fault model for the framework that we have designed.The system model shows how the various components fit together and validates the various design decisions that we have taken for our framework

Chapter 4

Algorithm

Algorithm 1 Working of Monitor Node

```
1: if NODE is a MONITORNODE then
2:   loop
3:     source = Receive(anysource)// listen for processor nodes
4:     if processblock(source) == NULL then
5:       Create_ Process_block()
6:     end if
7:     if MESSAGE = SETPIPELINE then
8:       // get next job from job list for processor
9:       next_job ← nextjobtobeexecuted
10:      for i = 0 to num_of_jobs do
11:        if taskgraph[i][next_job] ≠ −1 then //  if current job precedes next job
12:          // check previous job to be completed
13:          if job not complete then
14:            tryagain = 1
15:            break
16:          end if
17:        end if
18:      end for
19:      // next job is started after the last preceding job is complete
20:      start_time ← highest_prev_job_timestamp + next_job_releasetime
21:      if tryagain ≠ 1 then
22:        send(job_id,start_timestamp)
23:      else
24:        send("try again")
25:      end if
26:    end if
27:    if request = SETJOB then // a node finishes a task
28:      update(process_block)
29:    end if
30:    if time = receive_heartbeat_interval then
31:      receive("heartbeat",monitored_node_list)
32:    else if time = checkpoint_interval then
33:      send("checkpoint" ,monitored_node_list)
34:    else if time = receive_checkpointdata then
35:      receive("checkpoint data",monitored_node_list)
36:    end if
37:  end loop
38: end if
```

Algorithm 2 Working of Processor Node

```
1: if NODE is a PROCESSORNODE then
2:   loop
3:     if Pipeline is set then
4:       source = receive(pipeline) // receive task from pipeline
5:     else
6:       source = receive(monitor_node) // receive task from monitor
7:     end if
8:     if message = job then
9:       while timestamp ≠ start_timestamp do
10:        execute task
11:      end while
12:      send(monitor_node,task_info)// update process block
13:    else if message = try_again then
14:      wait(wait_interval) // wait for an interval before tryagain
15:    else if message = heartbeat then
16:      send(heartbeat_ack)
17:    else if time − heartbeat_receive = timeout then // heartbeat not received
       at proper time
18:      send(suspect_monitor_node)
19:    else if message = checkpoint then // checkpoint request
20:      send(checkpoint_data)
21:    end if
22:  end loop
23: end if
```

4.1 Algorithm Description

The interaction between various processor nodes is summarized in the following figure

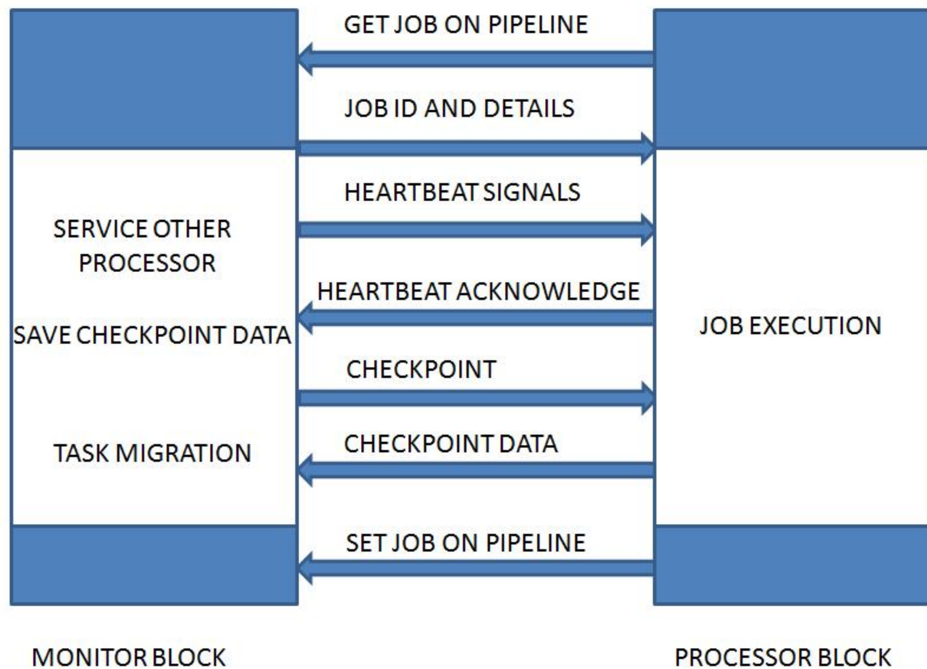

Figure 4.1: Processor Interaction Diagram

4.1.1 Types of Nodes

The nodes are mainly divided into 2 classes- monitoring nodes and processor nodes.

- Monitor Node They are completely connected to each other and to the processor nodes.They diagnose faults in processor nodes through heartbeats.They induce checkpointing to ensure proper fault recovery.Checkpointing is done in regular intervals of time as per the timestamp of the nodes.All the nodes of the system maintain a timestamp variable which is updated from a central clock and is used for synchronization.The monitor nodes also schedule the tasks on the pipeline when the event reset pipeline is fired by a processor node.This event is fired when the actual execution or release time of a task is different from the time given in the task graph maintained by the system.They are also responsible for migration of task execution from one processor to another when the node stops working due to a fault or overheating.Monitor nodes may appear to be a centralized system and may seem to present a single point of failure.However, the monitor nodes are made fault tolerant through the processor nodes.If the processor node does not receive a heartbeat signal to acknowledge, it suspects

the monitor node to be faulty and reports it to the other monitor nodes to replace.

- Processor Node The processor nodes are responsible for several functions such as residue generation,actuation of tasks and voting.The redundancy for each task is less than the number of monitor nodes used in the system.The analytical redundancy used in the processor nodes helps diagnose faults in a single iteration of the graph.In subsequent iterations, the faulty nodes are replaced which increases the reliability of each individual node and decreases the redundancy needed for each task.The processor nodes can determine faults in the monitor nodes on the basis of timeout in the acknowledgement to the heart beat signals they send to the monitor nodes.
  Important structures and variables include the process control block and the task graph. The task graph is represented by a sparse matrix or equivalent data structure like linked list or hashmap.it is defined as a matrix M of size $n * n$ where if $M[i, j] = k$ means i precedes j with a release time of k units.The process control block is depicted as shown in algorithm 2.It is maintained by the montoring system for each set of processors executing a task.It is implemented as a linked list,with each node representing a processor.
  It must be noted that structurally there is no difference between a monitor and a processor node,except that the monitor node must be completely connected to the processor nodes that they monitor.Due to this, a monitor node can be used to take the place of a faulty processor node in case there is a shortage of healthy processing units in the spare processor set.

4.2 Heartbeat and checkpointing mechanism

Checkpoints are taken by the processing nodes,the number of checkpoints depending on the type of task being executed.The processing node takes a checkpoint on receipt of a checkpoint signal from the monitor node.These checkpoint signals are also used as heartbeat signals to detect faults in the system as they are being sent regularly to the processing nodes by the monitor nodes.However as stated earlier,
$s = \Delta_i + l * \Delta_i - f * \Delta_f$
Hence,the monitor node starts sending additional heartbeats if it detects faults in processing units.This enables the heartbeat detection system to compensate for the decrease in redundancy of the task,because with increase in heartbeat signal frequency,the sampling duration decreases and the system is able to detect further faults in the system accurately.Thus the number of heartbeat signals being sent varies from zero to a maximum limit,after which the system becomes unsafe to continue operation.

4.3 Algorithm Analysis

Since our fault tolerance framework has been designed and implemented on a message passing system, our main parameter for analysis is the number of messages being

exchanged between the nodes and the resultant communication overhead.We assume that the communication overhead for an update message,which results in the update of the storage device,incurs more communication overhead than a simple query message.Thus the number of query messages carries less importance than the number of update messages.The execution occurs in the form of timeframes.The task graph iterates infinitely with each iteration being folded to subsequent iterations.The algorithm analysis has been divided into 2 parts-

Before setup of pipeline

if number of nodes=Nproc
number of tasks=Ntask
frame duration=L units
number of heartbeats=T per frame
x utilization of frame=H, with each frame containing P tasks
Numberofmessageswithnopipeline is given as
messages = $Nproc * Ntask + T * Nproc + Ntask$

After setup of pipeline

Numberofmessageswithpipeline is given as
messages= $Nproc * \frac{L}{2} - \frac{H}{2}$

4.4 Conclusion

This chapter provides the algorithm,its description and complete analysis .The working of the various kinds of nodes has been described in detail so that the functioning of the system is clear.The heartbeat and checkpointing mechnism been detailed here as it forms a very important part of the system.The complete algorithm has been divided into 2 parts for simplicity as well as to distinguish between the 2 major classes of nodes in the system.The processor control block has also been described and its various attributes have been outlined so that its role in the system can be fully grasped.

Chapter 5

Simulation and Results

5.1 Implementation

The various steps of implemetation are detailed below:

- Platform: In order to accurately portray a distributed embedded system for simulation purposes, vmware has been chosen running over windows vista.A vmware team is used,each of which runs on a minimal ubuntu installation described next. The team consists of a larger number of virtual emulations of ubuntu each of which represents 2 nodes as the underlying structure has 2 cores hence providing 2 processors. Ethernet adapters are used to connect the nodes to each other via LAN segments. The LAN segments can be set to have different extents of signal loss depending on requirement. Similarly, the LAN segments between the nodes can be changed to portray any topology required for simulation. There is a master node that is used to controll the simulation and is used for output of results .

- Operating System: Most distributed embedded systems use tinyos as the operating system of choice. For simulation process, a stripped down version of ubuntu 10.04 similar to is used. For this purpose, the latest version of the ubuntu kernel has been compiled and only the important services and daemons have been retained.The Ubuntu kernel was chosen because of its comparatively higher stability and familiarity.This reduced the minimum requirement of RAM for each node,resulting in formation of a larger number of nodes as is common in embedded systems. Only the simulation controlling node has a graphical user interface while the other nodes operate only on consoles to save memory.XFCE has been chosen as the GUI environment due to its minimal space and processing power requirements. From a simulation point of view the program execution is controlled by the node with GUI but from practical point of view all nodes are independent and controll is decentralised.

- Networking File System: The networking file system has been used to make sure that all nodes have the same versions of MPI libraries as the master node. The delays caused by NFS do not matter in the simulation as they are negligible for small clusters and DES which this project is meant to test at the current stage. In addition to NFS, openssh has been implemented to communicate between the nodes securely. The SSH has been kept passphrase free as the nodes are all parts of a distributed system and need full access rights to each other.

- Open MPI implementation : OpenMPI 2.4 has been installed in the master node as it is the latest version of OpenMPI implementation and is open source. All other nodes access openMPI installed on the master node via the networking file system described earlier. The program to be run is also loaded onto the master node which is then accessed by other nodes using NFS.

- Master Node: The master node is the node that is used to controll the entire simulation. However it is similar in computational functionality to the other

nodes and does not have any special status. It is the only node equipped with a graphical user interface.The program to be tested is also loaded onto the master node and the other nodes access it via NFS. The results of the simulation are also displayed on the master node's console.

- Graph Generation The analysis and result data derived from the openMPI system is used to generate graphs in a matlab environment.Matlab is also used to calculate the number of signals.

5.2 Simulation Results

With the above described simulation environment,the following graphs were obtained:

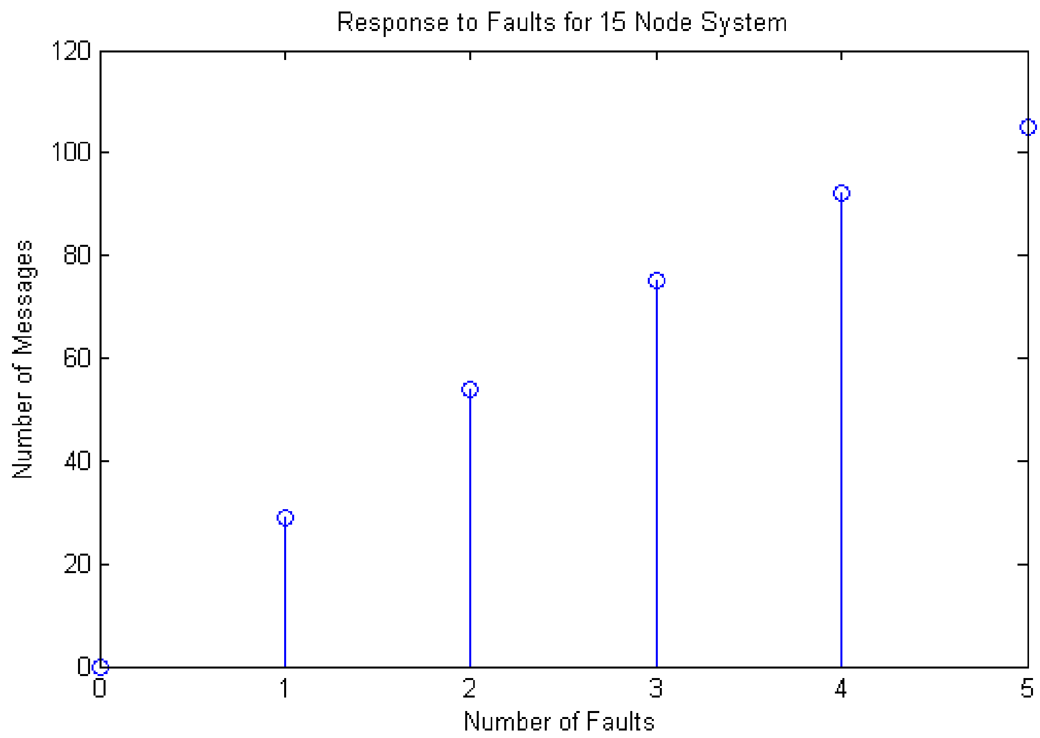

Figure 5.1: Fault Injection Simulation

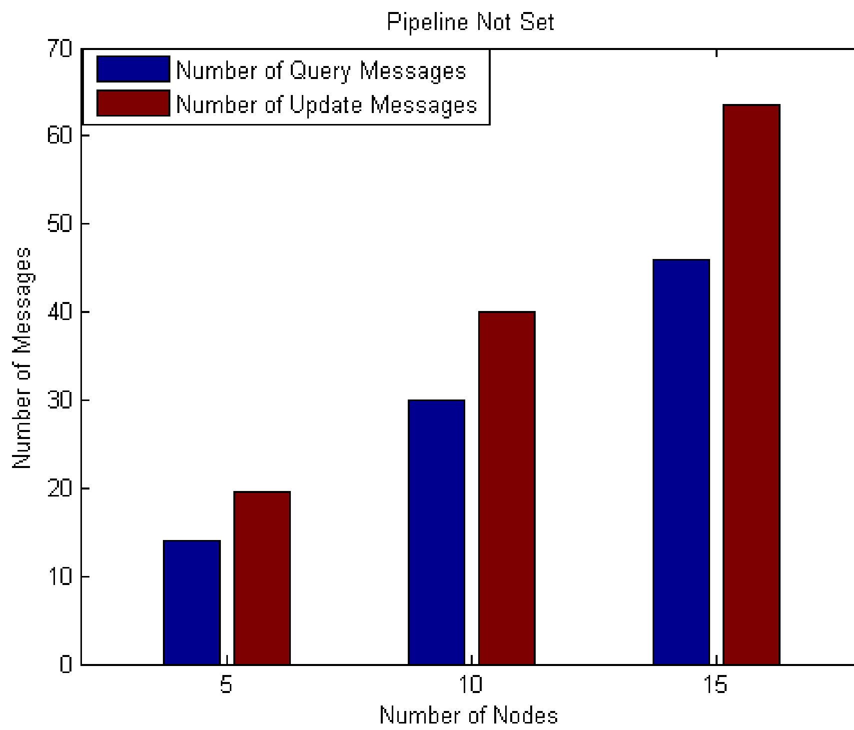

Figure 5.2: Messages passed with pipeline not set

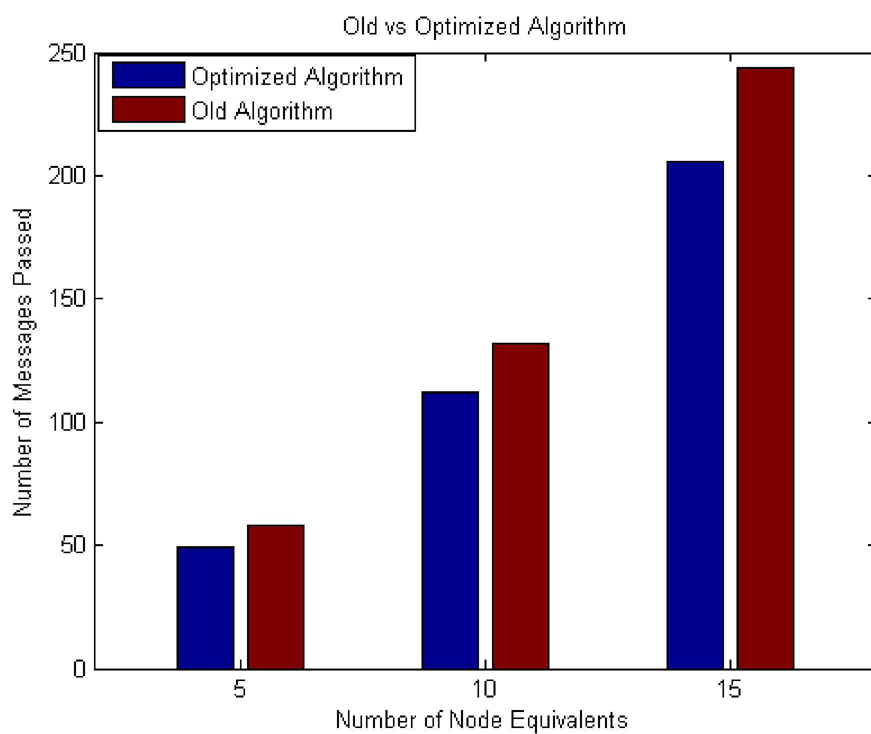

Figure 5.3: Optimized Algorithm vs Old Algorithm

5.3 Implementation Analysis

The results from the graphs agree with our algorithmic analysis.The graphs with no faults have a low message overhead due to "lazy" heartbeat monitoring,as described earlier..A fault injection graph has been provided that shows the change in heartbeat signals with the increase in number of faults.The graph is skewed due to the correspoding decrease in the number of healthy functioning nodes in the system.Overall the system succeeds in achieving its goal of reducing number of messages being transmitted in the system and also decreasing the total number of nodes used in the system while increasing the fault tolerance level of the system.

The system we have proposed results in a decrease of $N/2$ nodes with $N/4$ monitor nodes.This results in saving of $N/4$ over a traditional system using only redundancy.This also results in decrease in messages passed as shown in above figure 5.7

5.4 conclusion

This chapter provides us the simulation methodology used for the framework.We have also simulated the distributed embedded system using a completely new simulation method using openMPI.The results obtained are promising and prove our objectives to have been achieved.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis,We present a general,scalable framework for fault diagnosis in distributed embedded systems.Our framework achieves its goal of decreasing the number of nodes needed for fault diagnosis and increases the overall efficiency of the system.It also adheres to the deadline restrictions of the tasks,while limiting the number of messages being transmitted in the system.Fault diagnosis is also timely and fault recovery is done without the system falling into an unsafe state.Thus,we meet our aim of reducing the overall cost of the system while meeting the fault diagnosis and recovery requirements.

6.2 Future Work

The work presented in this thesis can be extended in a number of ways.The fault tolerance framework can be extended and re-designed for fly-by-wire systems,which comprise of far larger number of nodes than a steer-by-wire or brake-by-wire system. The framework can also be extended to include other types of faults such as byzantine faults.Finally, we can design a fault tolerance system for a wireless distributed embedded system in which there are no wired connectiions between the nodes and the nodes may be static or mobile.these are some of the fronts on which the work presented in this thesis may be extended

Bibliography

[1] “http://msdn.microsoft.com/en-us/library/dd129911.aspx.”

[2] “http://en.wikipedia.org.”

[3] N. V. R. Rajaraman, K. Ramakrishnan, Y. Xie, and M. Irwin, “Temperature and voltage scaling effects on electrical masking,” 2008.

[4] J. Council, “failure mechanisms and models for semiconductor devices,” JEDEC Publication, 2002.

[5] J. Gertler, “Fault detection and diagnosis in engineering systems,” Marcel Dekker,New York, 1998.

[6] N. Kandasamy, J. Hayes, and B. Murray, “Time-constrained failure diagnosis in distributed embedded systems: application to actuator diagnosis,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, pp. 258 – 270, march 2005.

[7] G. Goossens, J. Rabaey, J. Vandewalle, and H. De Man, “An efficient microcode compiler for application specific dsp processors,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 9, pp. 925 –937, sep 1990.

[8] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer, “A lazy monitoring approach for heartbeat-style failure detectors,” in *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pp. 404 –409, 2008.

[9] B. Randell, “System structure for software fault tolerance,” *IEEE Transactions in software.Engineering*, pp. 220 –232, 1975.

[10] Y. TAMIR and C. H. SEQUIN, “Error recovery in multicomputers using global checkpoints,” in *Proceedings of the International Conference on Parallel Processing*, pp. 32–41, 1984.

[11] F. Ghahfarokhi and A. Ejlali, “Schedule swapping: A technique for temperature management of distributed embedded systems,” in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pp. 1 –6, dec. 2010.