Static Scheduling in Real Time for Resource Optimization

Thesis submitted in the partial fulfillment of requirement for the award of degree of

Master of Technology

in

VLSI Design & CAD

Submitted by:

Ishan Khera

Roll No. : 601061012

Under the guidance of:

Mr. Ajay Kakkar

Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

JUNE 2012

DECLARATION

I, Ishan Khera, hereby certify that the work which is being presented in this thesis entitled "Static Scheduling in Real Time for Resource Optimization" by me in partial fulfillment of the requirements for the award of degree of Master of Technology in VLSI Design & CAD from Thapar University (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Mr. Ajay Kakkar.

The matter presented in this thesis has not been submitted in any other University / Institute for the award of any other degree.

Date: 26/06/2012

Ishan Khera
Roll No. 601061012

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Date: 26/06/12

Mr. Ajay Kakkar
Assistant Professor
ECED

Countersigned by:

(Dr. Rajesh Khanna)
Professor and Head ECED
Thapar University, Patiala
Date:

(Dr. S.K. Mohapatra)
Dean of Academic Affairs
Thapar University, Patiala
Date:

Acknowledgement

First of all, I would like to express my gratitude to **Mr. Ajay Kakkar, Assistant Professor**, Electronics and Communication Engineering Department, Thapar University, Patiala for his patient guidance and support throughout my work. I am truly very fortunate to have the opportunity to work with him. I found his guidance to be extremely valuable.

I am also thankful to our **Head of the Department, Dr. Rajesh Khanna,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department. I would also like to thank my friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

Lastly, I would like to thank my parents for their unconditional support and encouragement.

Ishan Khera

601061012

Abstract

Scheduling of tasks on the system is a decision making process which deals with performing functions like allocation of common resources, instruction sequencing, etc at specified intervals. Real time systems have become inseparable part of our lives and find many applications in different areas. The correctness of the results in real time systems depends not only on the logical computations but also on the time at which results are produced. A system needs to perform different tasks and scheduling of these tasks is done on the processor. It helps in the execution and completion of every task within desired time. For the efficient working of real time system it is necessary that all the tasks should meet their deadlines. The real time tasks can be broadly classified into three categories on the basis of deadline. The tasks which result in catastrophic conditions on missing the deadline are the hard real time tasks. The tasks whose results produced after missing the deadlines are of no use are known as firm real time tasks and those tasks whose results are useful even after missing the deadline are known as soft real time systems. The pre-emption is allowed in the real time systems and the overhead increases as the number of pre-emptions increases. Most of the embedded real-time systems only use necessary resources so that the extra pre-emption overheads among tasks should not degrade the system performance. The priority assignment schemes assign priorities to different tasks which can be static or dynamic. The study of various scheduling algorithms along with the comparison between different real time scheduling algorithms has been done in the thesis. In order to find a feasible schedule, a schedulability test is required which should ensure the meeting of deadlines. Resource optimization is of main concern in case of scheduling in real time. Shortest path algorithms achieve resource optimization by finding the shortest path between different nodes. Shortest distance between the nodes should be optimally evaluated which results in resource optimization and the tasks are scheduled through that optimized path in order to achieve feasible and efficient schedule.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

This chapter involves the scheduling and its need in real time operations. There is an utmost importance of scheduling tasks on the processors in order to achieve high speed and to minimize the overheads. The various techniques and parameters related to scheduling algorithms which are required for device optimization are covered.

1.1 Scheduling

Scheduling is an arrangement of performing functions like resource allocation, instruction sequencing etc. at specified time. The intervals between each function have been clearly defined by the algorithm to avoid any overlapping. Scheduling, in computing, means how the processes can be assigned on the available CPU. It is a key concept of multitasking, multiprocessing and real-time operating system design. It is done by a means of scheduler and dispatcher. A *scheduler* is a person or machine that organizes or maintains schedules. In computing, scheduler is software program that arranges jobs or computer's operations in an appropriate order. A *dispatcher* is a module which gives control of CPU to the process selected by the scheduler.

1.2 Need of Scheduling

Scheduling is a decision making process which deals with the allocation of common resources to various tasks at different time periods for the achievement of multiple objectives. The resources and tasks can be of different forms in homogeneous/heterogeneous organization. Priorities have been associated with the tasks; each task has its due date and earliest dead line. Similarly objectives can also be in the different forms, they can be minimization of completion time of last task or minimization of number of tasks completed after their respective due dates [4,8-9]. In an industry it may mean to assign appropriate workers to do some task each day. It enables us to perform tasks which are done in routine automatically and efficiently.

1.3 Performance Measures of Scheduling

There are many performance measuring parameters in the scheduling technique. The following parameters need to be achieved in order to schedule the task perfectly on a given processor.

- **Fairness**

It is important goal to achieve under all circumstances. A scheduler guarantees that each process should get a fair amount of CPU time for its execution and there should be no condition of starvation. But giving equal or equivalent time to all the processes is also not fair as different processes are not equally critical.

- **Throughput**

It is the amount of work a computer can do in a period of time. The scheduler aims to maximize the number of jobs completed or processed per unit time.

- **Turnaround Time**

It is the amount of time taken for the process to get executed. It is the interval from the time of submission of a process to the time of completion of the process.

Turnaround time, $T_r = T_s + T_w$

T_s = Execution Time, T_w = Waiting time

The scheduler should minimize the turnaround time, but it is not upto the scheduler to affect the execution time but it can minimize the waiting time so as to reduce the turnaround time [3,8].

- **Waiting Time**

Similar processes are allocated equal CPU time or times are divided according to processes' priority. The time every process waits for its execution in the ready queue is known as waiting time and it is the duty of scheduler to minimize this time [22].

- **Efficiency**

It signifies the amount of time the system is performing tasks. Scheduler should keep the system busy for most of the time when possible. If both CPU and I/O devices are working all the time then more work gets done per unit time [24]. Usually the above goals conflict with each other, as amount of the CPU time is finite, so there is a need of trade-off between the achievable goals. Every scheduling technique aim to achieve either of above mentioned goals. For scheduling tasks on a processing unit, there can be an optimization for power along with clock period. There are various techniques devised to optimize power and processing speed.

1.4 Scheduling Algorithms

It is a method by which resources are allocated to the processes such as bandwidth, processor time etc. It is usually done to distribute workload across multiple computers or processing units so as to achieve optimal resource utilization, maximize throughput, minimize response time and avoid overload [26]. The goal of scheduling algorithm is to fulfil the following criterion.

- **No Starvation**

This means that a particular process should not be held up indefinitely. There should be proper allocation of resources to every process so as to ensure that all processes get CPU time [25].

- **Pre-emption in case of priority based algorithms**

Scheduling algorithms have to ensure that there should be fairness in the pre-emption policy [26]. It should make sure that high priority tasks should not hold low priority tasks indefinitely.

- **Good scaling as the number of tasks increases.**

As the number of processes increases, it becomes inefficient to compute dynamic priorities. Scheduler in the Linux kernel computes the priorities only when all runable processes have exhausted their time quantum [24].

1.5 Categorization of Scheduling Algorithms

Algorithms are classified on the basis of different parameters. The classification can be based on the type of application for which scheduling needs to be done, whether the tasks need to run time schedule or compile time schedule, whether the scheduling is done on central site or distributed sites, uniprocessor/multiprocessor scheduling.

1.5.1 Classification from User's Point of View

From the user's point of view, scheduling algorithms are classified into 3 categories.

- **Iterative Scheduling**

Scheduling of processes is done iteratively and the algorithms used for scheduling are known as iterative scheduling algorithms [26]. Round Robin, shortest process next, lottery scheduling etc are the examples of iterative scheduling algorithms

- **Batch Scheduling**

Processes are queued together in a batch and scheduling is done in batches. Algorithms that are used to schedule the batches are known as batch scheduling algorithms [26]. FCFS, Shortest

remaining time next, highest response ratio next are the examples of batch scheduling algorithms.

- **Real time scheduling**

Real time tasks are those in which the accuracy of the outcome not only depend on the correctness of result but also depend on the time at which the results are produced. Scheduling such tasks is done by real time scheduling algorithms [23]. Rate monotonic and Earliest Deadline First (EDF) are examples of real time scheduling algorithms.

1.5.2 Classification Based on the Time of Schedule

This classification is done on the basis of time of scheduling the processes i.e. whether the processes are to be scheduled on the compile time or run time.

- **Static Scheduling**

In this technique, scheduling decisions are made at compile time. For scheduling, complete prior knowledge of task-set characteristics is required. The system's behaviour with static scheduling is deterministic [22]. Rate monotonic scheduling is the example of static scheduling used for scheduling real time tasks.

- **Dynamic Scheduling**

Scheduling decisions are made at run time by selecting one task out of the set of ready tasks [22,26]. Dynamic schedulers are flexible but also require run time in finding a substantial schedule. System's behaviour is non-deterministic. EDF is the example of dynamic priority scheduling algorithm used to schedule real time tasks.

1.5.3 Classification Based on the Site of Scheduling

This classification is based on whether the processes are to be scheduled on a central site or on distributed sites.

- **Centralized Scheduling**

All decisions are made at the central site [22]. The central scheduler in the distributed system is a critical point of failure. Updated load situation information is needed on all nodes which might lead to communication bottleneck.

- **Distributed Scheduling**

Scheduling of non-interactive processes or jobs in a network of computers comes under this category [19]. It refers to the chaining of different jobs into a coordinated workflow that spans several computers.

1.5.4 Classification Based on Pre-Emption

Pre-emption means prediction of higher priority task. Depending upon, whether pre-emption is allowed or not, scheduling algorithms can be classified into two categories.

- **Pre-emptive Scheduling**

If the system can be interrupted during the execution of the process on the arrival of higher priority task then this type of system is known as pre-emptive system and scheduling algorithms used to schedule such systems are known as pre-emptive scheduling algorithms [28]. All real time scheduling algorithms are examples of pre-emptive scheduling algorithms.

- **Non Pre-emptive Scheduling**

If no interruption is allowed during the execution of process then scheduler is known as non pre-emptive scheduler [19]. First Come First Serve (FCFS) scheduler is non pre-emptive scheduler.

1.5.5 Classification Based on the Number of Processes to be Scheduled

This classification is done considering whether the scheduling is done on single processor or multiple processors.

- **Uniprocessor Scheduling**

If the scheduling is done on a single processor then it is known as uniprocessor scheduling [22]. Round Robin, RM scheduling etc are the examples of uniprocessor scheduling algorithms.

- **Multiprocessor Scheduling**

If number of events occurring close together is high then we have to increase number of processors in the system. Such system is known as multiprocessor systems and scheduling techniques required to schedule a task on such system are known as multiprocessor scheduling algorithm [19,31]. Global scheduling algorithms and partitioning scheduling algorithms fall under this category.

1.6 Scheduling Issues

While performing scheduling, there are multiple constraints which are to be met for accurate scheduling. There are issues relating to different tasks the system performs. The requirement of the system decides the scheduling algorithm design [25]. We have to consider the application profile of the system; the level of scheduling required, the time at which a process is scheduled, the parameter to be optimized, the need of pre-emption etc. Some of the issues are classified below:

- **Application Profile**

It specifies the type of applications to be run on the system. A program alternates between CPU utilization and I/O. While performing scheduling we need to consider the bound of the programs executed on the system [24]. It can be either compute-bound or I/O bound. For the I/O bound operations scheduler decides which processes should be scheduled while the currently running task is performing some I/O operation.

- **Scheduling level**

There can be only one process at a time which gets CPU time. So, the scheduler has to decide the process which is to be executed next. The decision is dependent on the choice of scheduling algorithm. Swapper is provided to decide about the processes which should reside in the memory scheduling and is known as midterm scheduling.

- **Time of schedule**

It is the duty of scheduler to decide about the time at which schedule is started. This is done by the help of scheduling algorithm. The response of the system on receiving an interrupt, on creation/termination of a process, etc is determined by the scheduler [27]. When there is a system call, it is the scheduler which decides which processes are to be put in ready queue.

- **Pre-emptive or Non pre-emptive**

The system can be interrupted during the execution of the process in case of arrival of a higher priority process. This type of system is known as pre-emptive system [19]. If no interruption is there then the system is non pre-emptive system.

- **Parameter to be optimized**

There are different parameters associated with the systems and we can optimize some of them depending upon the requirement of the system. As optimization of one parameter may affect the other parameter so all the parameters cannot be optimized. Fairness should be achieved for all

types of systems. Batch systems require higher throughput whereas interactive should have minimum response time [27]. Other parameters that need to be optimized can be waiting time, turnaround time, CPU utilization, etc.

- **Scheduling overhead**

Every scheduling technique incurs some overhead. There are many context switches which make the system expensive and have to be minimized.

1.7 Motivation

Real time systems are the computing systems that must react within the precise time constraints to events in the environment. The efficient scheduling of tasks on these systems becomes necessary in order to receive accurate and timely response. From the work done by various researchers it has been observed that there is still a need to optimize scheduling techniques, so that, the various constraints of real time systems can be met and performance can be enhanced.

1.8 Objective of the Thesis

The main objective of my work is to achieve resource optimization and resolving timing constraints in scheduling the task in real time.

1. To study various algorithms used for scheduling processes on a processor. The working of the scheduling techniques in different environments is to be studied.
2. To study the real time systems and their application domains. The study of various techniques used for scheduling tasks in real time and the comparison of those techniques.
3. Evaluation of shortest path algorithm with minimum delay and less probability of error along the particular path.

1.9 Organization of Report

The report has been divided into five chapters which contain various scheduling techniques and the introduction to real time systems. The first chapter involves the basic introduction of scheduling, its need, and various performance measures of scheduling. The classification of scheduling and the various scheduling issues have also been discussed. Chapter two contains the literature survey and the various scheduling techniques used to schedule the tasks on different systems. Third chapter has the introduction of real time systems and their classifications. The known real time scheduling algorithms has been explained and compared. The results and discussions on the basis of real time scheduling for the evaluation of shortest path have been shown in forth chapter. Finally conclusions and future scope of the work has been presented in fifth chapter.

Chapter 2 Literature Survey

This chapter has the survey of the work done by various researchers in the field of scheduling tasks in real time. The possibility of improvement in the work and motivation has been stated clearly. There is also a brief description of some of the scheduling algorithms which is used for scheduling tasks in different systems.

2.1 Literature Survey

The following sections show the work done by the various researchers in the field of scheduling for real time processors and also the various scheduling techniques known.

Sanjoy K. Baruah et. al.[1] presented a schedulability analysis of Rate Monotonic (RM) scheduling algorithm to determine schedulability of periodic task set on a particular uniform multiprocessor. They considered periodic model of hard real time tasks with two parameters namely execution parameter (C_i) and period (T_i). They have provided the conditions of feasibility of the task system on a uniform multiprocessor platform. Also they proved that all the deadlines will be met if task set is scheduled using Rate Monotonic (RM) algorithm.

Peng Li et. al. [2] presented two fast and best-effort scheduling algorithms for real time systems. The algorithms are known as Modified Dependent Activity Scheduling Algorithm (MDASA) and Modified Locke's Best Effort Scheduling Algorithm (MLBESA) which are similar to Dependent Activity Scheduling Algorithm (DASA) and Locke's Best Effort Scheduling Algorithm (LBESA) respectively. A benefit function is associated with each task which specifies the benefit upon the completion of task. The algorithms removed serious bottlenecks for resource allocation algorithms. Thus, they can be used by resource allocation algorithms such as **R**esponse-Time-Based **B**est-Effort Resource **A**llocation (RBA) for determining the response times under DASA and LBESA in a faster way. The computational complexity of RBA is reduced which resulted in a good performance at low computational cost. MDASA and MLBESA heuristically, yet accurately mimic the behaviour of DASA and MLBESA respectively but are faster with *O(n)* and *O(nlog(n))* worst-case complexities, respectively. Also, under overloaded conditions DASA and LBESA outperform MDASA and MLBESA respectively. The

algorithms were not suitable for overloaded systems and resource allocation cannot be done efficiently in overloaded conditions.

Xiaoying Wang et. al. [3] has proposed a method to analyze and improve the pre-emptive behaviour of Priority scheduling algorithms. They analyzed accumulate waiting time, executing time, deadline etc of the tasks in the ready queue. They presented a waiting time limit formula of each task in the ready queue while guaranteeing its deadline. The longest postponement time and final pre-emption time of higher priority task in the ready queue was reduced which further decides whether to pre-empt or not. The dependency of the pre-emption relations was not only on task's priority but also on the time to reach its deadline. Micro scheduling of the tasks in the ready queue was done in order to reduce the extra overheads like processor, storage and stack resources. Also, the pre-empt amount of task set was decreased. They showed that micro scheduling algorithm decreased the number of pre-emptions from 0% to 86.4% which avoided unnecessary pre-empt overheads to a large extent. In addition, micro scheduling algorithm also improved processor schedulable utilization of Rate Monotonic (RM), namely enhanced reliability and real time performance of system. The decrease in the pre-empt amount is not sufficient enough and complexity of the system has increased.

Chen Tianzhou et. al. [4] proposed a real time scheduling algorithm for embedded systems with various resource requirements. The performance of the scheduling algorithm affects the performance of whole system. Hard real time embedded operating systems needs very rigid response times for better performance. They divided all the tasks into sets and scheduled the tasks according to the features of the set. There are three sub-procedures in the procedure of algorithm, namely, i) Set Division, ii) Scheduling inside one set, and iii) Scheduling among sets. The tasks are divided into different sets on the basis of the resources of the embedded system which are finite in number. The partitioning of the procedure added an overhead and the performance of the system were not optimized. The energy constraint has not been taken care of.

Jiajing Zhuo et. al. [5] presented a task scheduling algorithm for a single processor parallel test system. They implemented a parallel test in the single processor auto test system and put forward a heuristic parallel task scheduling algorithm: Scheduling-Q which can meet the characteristics of the auto test system. In test system some test tasks can be executed with different resource

allocations. The algorithm schedules the test tasks according to the task's earliest starting time and the test generalized resource loading. The generalized resource loading is incorporated as task resources set loading based on resources allocation mode and task resources set loading based on task's starting time. The test resources with bigger loading have more opportunities to obtain task and are always in a busy state. Thus resources loadings can be balanced to a degree. So the parallel performance of test system has improved with the algorithm. In addition, the algorithm adopted the strategy of heuristic local optimum search and thus the time complexity of the algorithm is decreased. The priority assignment of the tasks is not optimal as it may lead to the busy state all the time and condition of deadlock may arise.

S. Roman et. al. [6] presented constant complexity scheduling for hardware multitasking in two dimensional reconfigurable field-programmable gate arrays which described the need of extending operating system to manage the FPGAs. An algorithm has been presented which decides the scheduling and placing of arrival tasks with real time constraints, like deadline, in FPGA device. The division of FPGA is done in four parts with different sizes and each incoming task was placed in one of these partitions depending upon its size and parameters. It was also possible to make changes in the size of partition, number of partitions, queue selection policy etc. at the run time. The results proved that their four partition configuration is equally or more effective than First Fit (FF) algorithm. Even in the worst case their algorithm is better than FF, for eg., in case of pJPEG.

Pravanjan Choudhary et. al. [7] presented hybrid scheduling of dynamic task graphs with selective duplication for multiprocessors under memory and time constraints, which describes the scheduling methodology for the task graphs to embedded systems with multiple processors. Methodology was presented in three phases which was designed for task graphs that were dynamic in nature due to the presence of conditional tasks and tasks those were unpredictable and bounded. The nodes were mapped in the first phase and the critical nodes were identified in the second phase. Those critical nodes were duplicated for the possible rescheduling at runtime. The third phase is the online phase which performs runtime scheduling. The experiments indicated that the methodology proposed is suitable for task graphs that have higher number of conditions, a higher parallelism, and a significant non determinism in the execution time of its nodes. It was shown that computation time of the static phase and overhead increases with

increase in number of nodes but still remained in acceptable limits. Overall the scheduling overhead stayed in microseconds which were very low as compared to conventional techniques. The complexity of the system was increased with the additional overheads of online strategy. The ability to find shortest path for the transmission of data is the area of improvement.

Euiseong Seo et. al. [8] presented an energy efficient technique for scheduling real time tasks on multicore processors to lower the power consumption and increasing the throughput. They presented two techniques which modify existing techniques of unicore processors for multicore processors. The two techniques suggested by them are: (i) Dynamic Repartitioning algorithm, which dynamically balances the task loads multiple cores to minimize the power consumption during execution. (ii) Dynamic core scaling algorithm, which reduces leakage power consumption by adjusting the number of active cores. The simulation results show that 25% of energy consumed can be conserved by dynamic repartitioning and 40 % is conserved by dynamic core scaling. The possibility of dependency of the task set was not considered and the complexity of dynamic partitioning was of major concern.

Ming Xiong et. al. [9] presented the deferrable scheduling algorithm which maintains real time data freshness for minimizing the update workload by maintaining the temporal validity of real time data. The update transaction follows aperiodic task model and sampling times of update transaction jobs were deferred as late as possible. Aperiodic task model is adopted and the necessary and sufficient condition for schedulability of the algorithm is not clear. Only static priority techniques are considered and behaviour with dynamic priority algorithms is unknown.

Fengxiang Zhang et. al. [10] presented schedulability analysis for real time systems with EDF scheduling which reduces the number of calculations of the processor demand of every task set and thus reduces the calculation times for the schedulable as well as unschedulable task sets. Experimental results show that 96% of task sets complete each schedulability test in less than 30 calculations of processor demand function ($h(t)$).

Enrico Bini et. al. [11] presented a response-time bound in fixed-priority scheduling with arbitrary deadlines in which they indentified the desirable properties of estimates of the exact response time. Repeated computation of the worst case response times slows down the system

which is undesirable for real time applications. They proposed a technique which possessed the properties which are continuity with respect to system parameters, efficient computability and approximability for the estimation of worst case response time of sporadic task systems which are fixed priority scheduled on a pre-emptive uniprocessor. They have derived the continuous upper bound on the response times for task systems which are to be scheduled on the pre-emptive processor and proved that the exact response time should be as large as this bound if the system is implemented on the processor which is at most only 50% as fast. There is loss of accuracy with the continuity of the response time bounds of the task systems.

Marko Bertogna et. al. [12] given the schedulability analysis of global scheduling algorithms on multiprocessor platforms in which they had addressed the problem of pre-emptive scheduling of periodic and sporadic task sets with constrained deadlines on a multiprocessor platform. They had assumed that the global work conserving scheduler with migration possibility of task from one processor to other was there. The analysis had been applied to fixed priority and Earliest deadline first scheduling techniques and the schedulability conditions thus derived were tightened which resulted in significant improvement in percentage of the accepted task sets. The schedulability algorithm presented could check whether a periodic or sporadic task set could be scheduled on a multiprocessor platform in polynomial or pseudo polynomial time. The procedure "*Schedulabilitycheck*" could detect the maximum number of schedulable task sets among all existing task sets. Resources were shared which results in blocking time when those shared resources are accessed exclusively and evaluation of the blocking time is necessary.

Wann-Yun Shieh et. al. [13] suggested energy efficient tasks scheduling algorithm for the dual core real time systems to satisfy the low power consumption demands. Their main aim was to minimize the power consumption at the same time they maintain the same performance level of the dual core processor. They include two online and offline attacks and proposed an Integer linear programming approach for the optimal scheduling. The experimental results shown in their work was evident that the energy consumption was could be about 38% effectively by using heuristic algorithm. The two functions used increases the complexity of the system and the proper resource utilization is not achieved. The resource utilization needs to be improved so as to get enhanced performance.

Marko Bertogna et. al. [14] suggested limited pre-emption earliest deadline first scheduling of sporadic task systems. Their algorithm is based on the amount of pre-emption. With no pre-emption there will be significant scheduling overhead and as each pre-emption leads to context switching, it causes increase in runtime overhead. In their paper, they suggested a limited pre-emption technique which can schedule all the systems that can be scheduled by fully pre-emptive algorithm but with limited runtime overhead. It is provided with a non pre-emption function, Q, which is monotonically non-increasing. It takes time to the deadline of executing job as input and gives us the time for which such job could be executed non-pre-emptively. They performed large number of simulations by varying number of tasks, n, total utilization, U, and task parameters. For $n = 10$ the average number of pre-emptions has been reduced by 20% as compared to normal EDF. Also, it had been observed that with the increase in number of tasks, keeping total utilization constant, the average number of pre-emptions remained almost constant. The limited pre-emption technique used adds an additional overhead as compared to static priority techniques where there is no overhead in terms of calculating priorities. The performance is enhanced as compared to fully pre-emptive technique but Worst Case Execution Time (WCET) includes the pre-emption overhead. So, there is a need to evaluate the pre-emption overhead.

Xian-Bo He et. al. [15] suggested an improved version of earlier deadline first (EDF) scheduling algorithm which was based on Fuzzy Inference System (FIS). Their algorithm was suitable for embedded real time systems in uncertain environments in order to minimize the deadline miss ratio. All the tasks were periodic and fuzzy set describes the task's criticality and deadline distance. The tasks which have higher criticality and shorter deadline distance have the higher priority and are scheduled first. The experimental results shown in their paper were evident that the total deadline miss ratio was reduced compared to traditional EDF. Important tasks in improved EDF never miss their deadlines when the system is not overloaded but the it is not suitable for hard real time tasks.

A. Burns et. al. [16] suggested partitioned EDF scheduling for multiprocessors using a new task splitting scheme. They have compared $C = D$ splitting scheme with fully partitioned scheme. They suggested that for m processors there can be splitting of at most m-1 tasks. No special run time mechanisms were required and overheads were kept minimal. In fully partitioned systems,

there is a problem of bin-packing due to overloading of processors. The suggested scheme was straightforward to implement without any unusual real time operating system functions. They considered that each processor used standard EDF policy and tasks have dynamic priorities. Evaluation of the technique showed that there was a remarkable increase in the performance. The average utilization factor of full processors with 8 tasks and with a total utilization of 4 had increased from less than 0.89 to 0.99. The policy needs to be evaluated for the task sets with arbitrary deadlines and the comparison with other EDF based task splitting approaches. A utilization bound for this scheme was not derived and for high utilization the study of pre-allocation schemes is necessary.

Wan Yeon Lee [17] proposed energy-efficient scheduling of periodic real-time tasks on lightly loaded multicore processors which considered the processors contained more processing cores than running tasks and had dynamic voltage and frequency scaling capabilities. The energy saving techniques are introduced which were turning off the rarely used cores and exploiting the overabundant cores for executing the task in parallel with lower frequency. It had been verified if the techniques were supported then the problem of minimizing energy consumption of real time tasks was reduced to NP hard and deadlines were also met. A polynomial time scheduling scheme was also proposed that provided a near minimum energy feasible schedule. The evaluation results showed that the proposed scheme saved up to 64% of the energy consumed as compared to the other schemes which considered the execution of each task on a separate core.

Motivation from Literature Survey

From the literature survey, it has been clear that a lot of work has to be done in the field of scheduling tasks in real time in order to reduce the delay between the nodes. There are some areas where the improvement can be made like the resource utilization can further be optimized while scheduling in real time. In order to achieve resource optimization and enhanced performance, there is a need to work on the graph theory and shortest path algorithms which can result in optimized usage of resources.

The various scheduling techniques that exist are described below:

First Come First Serve (FCFS)

It is the simplest scheduling algorithm in which processes are dispatched according to their arrival time on the ready queue. It is a non pre-emptive technique, so when processes get the CPU time, they are executed to completion. It is fair in the human sense of fairness but unfair in the sense that long jobs make short jobs wait or important jobs might get held up because of unimportant jobs. As the process is executed until it gets completed, there is no condition of starvation as long as process completes its execution [23]. No prioritization of processes is there. It cannot guarantee good response time, so this technique is inappropriate for interactive systems. It is rarely used in modern operating systems but it is usually embedded within other schemes.

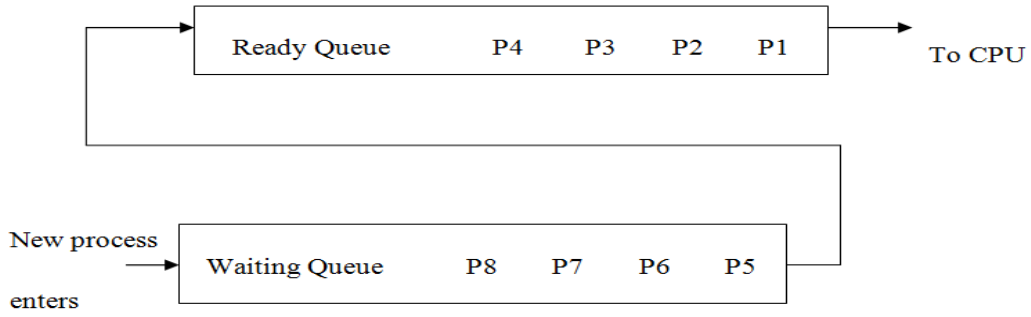

Figure 2.1: Scheduling processes using First Come First Serve (FCFS) technique

Consider there are 5 processes namely, A, B, C, D and E which are to be scheduled using FCFS algorithm.

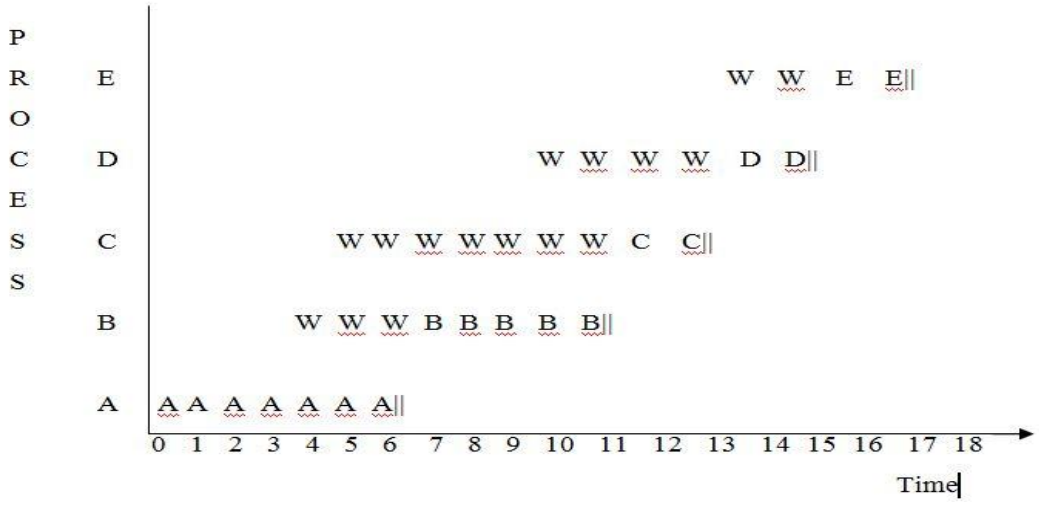

Figure 2.2: Example of FCFS scheduling

T= elapsed time, including waiting time

t= required processing time

M= T-t = missed time

R= t/T = ratio

P = T/t = penalty rate = 1/R

Process	Arrival time	Time required, t	Time elapsed, T	Missed time, M	Ratio	Penalty
A	0	7	7	0	1	1
B	4	5	8	3	5/8	8/5
C	5	2	9	7	2/9	9/2
D	10	2	7	5	2/7	7/2
E	14	2	5	3	2/5	5/2

Table 2.1: Scheduling Table using FCFS Technique

Round Robin

It is the simplest algorithm for pre-emptive scheduler. In order to fairly schedule a process, round-robin scheduler employs time-sharing technique. A time quanta is assigned to each process in equal portion without any priority to any process. The selection of time quanta is of main concern. The larger the quantum, the slower is the response and the shorter the quantum, the larger portion of time is wasted in context switching. It is easy to implement and starvation-free. Average waiting time under this policy is very large [23].

Consider the same problem of 5 processes as in case of first come first serve scheduling, apply round robin scheduling, the following schedule is obtained

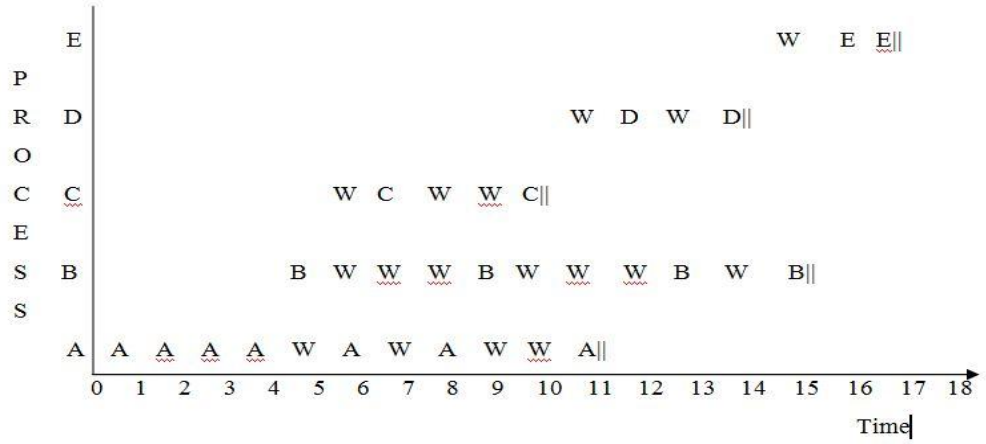

Figure 2.3: Example of Round Robin scheduling

Process	Arrival time	Time required, t	Time elapsed, T	Missed time, M	Ratio	Penalty
A	0	7	11	4	7/11	11/7
B	4	5	11	6	5/11	11/5
C	5	2	5	3	2/5	5/2
D	10	2	4	2	2/4	4/2
E	14	2	3	1	2/3	3/2

Table 2.2: Scheduling Table using Round Robin Technique

Priority Based Round Robin

This type of scheduling is similar to round robin technique but it allows hierarchy of processes. It is also known as multilevel queue scheduling. The processes are assigned to different queues on the basis of some property of the process. Waiting time and turnaround time depends on the priority of the processes. It is simple to implement and have reasonable support for priorities. Starvation is very rare as almost all the privileged processes are blocked for I/O.

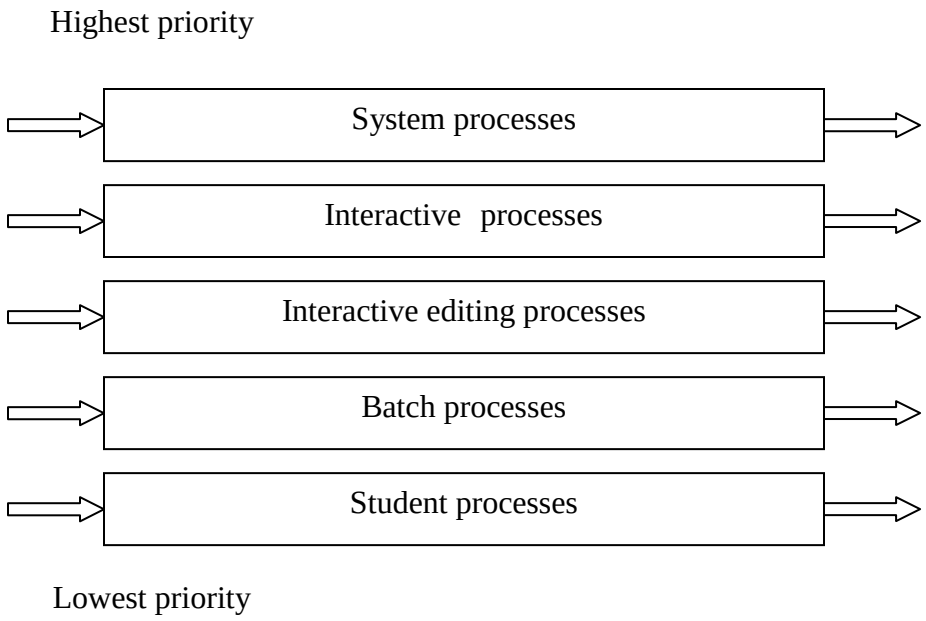

Figure 2.4: Scheduling using Multilevel Queue technique

Shortest Process next

In this scheduling policy, scheduler schedules the processes with the least estimated processing time to be next in queue [22]. This technique is also known as Shortest-Job-First or shortest next-CPU-burst first. CPU burst is the time processor is being used by process before it is no longer

ready or how long a process requires CPU between I/O waits. This policy considers that we know the next CPU burst of all the ready processes. The estimation of the length of next CPU burst is based on the recent CPU bursts. This technique is non-pre-emptive but pre-emption can be included with this technique and the resulting technique is known as Shortest Remaining Time next. The problem with shortest job first algorithm is that it cannot handle infinite loops and delivers poor performance if the task with short CPU burst comes after the process with longer burst which has started execution. Another problem associated with this technique is starvation i.e. processes with long burst times are indefinitely postponed from getting on the processor. In most of the cases, this technique is designed to achieve maximum throughput as short processes get most of the CPU time.

Lottery Scheduling

It is the probabilistic scheduling algorithms for the processes in the operating systems. Each process is assigned some number of lottery tickets and priority is determined by the number of tickets each process has. Lottery tickets represent resource rights and allocation is determined by holding a lottery [20]. The priority is relative percentage of all the tickets competing for the resource. The winner is picked randomly by the scheduler and resource is allocated. Tickets can be used for variety of different uniform resources and are machine independent. If the client has p probability of winning then expected number of wins is given by np. This technique is accurate most of the times but short-term inaccuracies may occur. It solves the problem starvation by allowing pre-emption.

Shortest Remaining Time

This is the pre-emptive algorithm where pre-emption is based on the process' next CPU burst time. The currently running process is interrupted on the occurrence of process with shorter processing time. This is an efficient technique for shorter processes but delivers bad performance in case of long processes [24]. It can also be categorized as priority based algorithm with priorities to the processes with shorter processing time. The pre-emption is done on basis of next process' CPU burst and its exact estimation is not easily possible, so it is used in special environments only where estimation can be possible.

Highest Response Ratio Next

This technique is developed by Brinch Hansen to correct certain weaknesses in Shortest Job First algorithm including the difficulty in estimating the run time. It is an non-pre-emptive technique in which processes cannot be interrupted in between their execution. It is the priority based

algorithm in which priority is dependent on the estimated run time and also on the waiting time of the process. Jobs with longer waiting time get higher priority and hence the problem of starvation is removed.

$$\text{Priority} = \frac{Waiting\ \ time\ \ + Estimated\ \ run - time}{Estimated\ \ run - time}$$

Multilevel Feedback Queue Scheduling

This type of scheduling was first described by Corbato et al. in 1962 in Compatible Time-Sharing System (CTSS). This algorithm aims to minimize turnaround time along with reducing the response time. The idea is to separate processes with different CPU-burst characteristics. In this technique there are multiple queues with different priorities. The job in the higher priority queue is executed first and the jobs within the same queue are scheduled using round robin algorithm [19]. The priority is varied on the basis of observed behaviour. A new process in the system is placed in the highest order priority queue and if uses entire time interval during execution then the priority of that process is reduced and it is dropped by level in the priority queues. If it completes its execution before the end of time interval then it remains at the same priority level. First come first server (FCFS) technique is used in all the queues but the bottom level queue or the lowest priority queue uses Round robin technique. I/O-bound processes will stay in higher priority queues. With fixed quantum time, the turnaround time of longer processes can stretch out. To fix this we can increase the time quantum with the depth of the queue or processes can be promoted to higher level queues after some time.

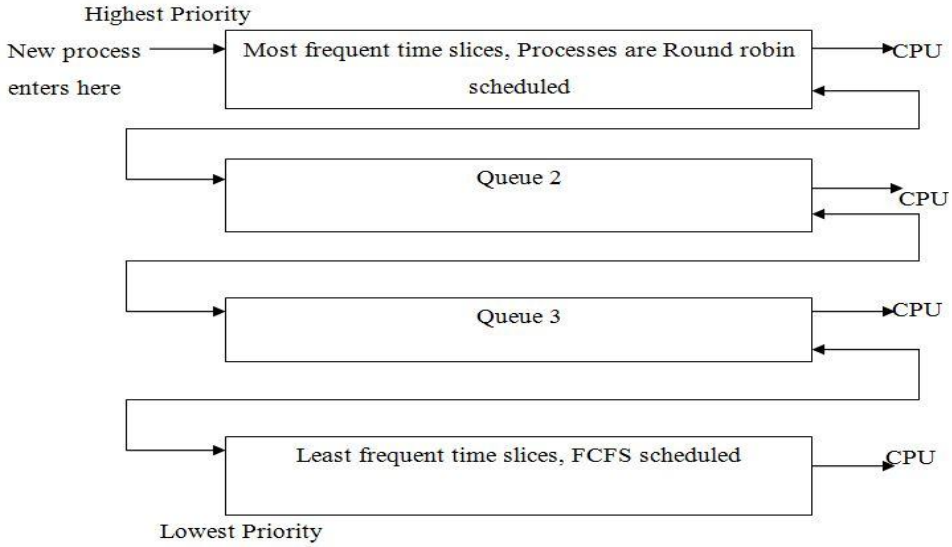

Figure 2.5: Process scheduling using Multilevel Feedback Queue Scheduling

Rate Monotonic Scheduling (RMS)

This algorithm was presented by Liu and Layland in 1973. It is a dynamic pre-emptive scheduling algorithm used to schedule real-time threads with a guarantee that none of the thread will ever miss its deadline. Each task is assigned priority according to its interval; task with shortest interval gets highest priority, hence the name "Rate Monotonic". The priorities are assigned to the tasks statically. This technique is used for the multiple processes to be run on a single processor [19]. A task set is considered schedulable if all tasks meet all deadlines all the time.

Earliest Deadline First (EDF)

It is a dynamic priority real time scheduling policy. The processes are placed in priority queue. A deadline is assigned to every task and the task with closest deadline is executed first. In order to implement this, the deadline of the task and amount of time required to do the task must be known. EDF can guarantee all deadlines are met if CPU utilization is not more than 100%, so compared to RMS EDF can guarantee all the deadlines in the system at higher loading [25]. In overloaded systems, the set of processes which miss deadline are highly unpredictable and it is also difficult to implement in hardware and also there is tricky issue of representing deadlines in different ranges. This is why it is not found in industrial real time computer systems.

2.2 Types of Operating System Scheduling

Basic operating systems schedulers can be of 3 distinct types. Scheduling types are divided according to the schedulers and the names of schedulers suggest the relative frequency with which these functions are performed [23]. The description of the schedulers is given below.

2.2.1 Long-term or high-level scheduling

It decides which processes are to be added to the set currently executing processes and which are to be exited. It controls the degree of multiprogramming in multitasking systems with a need of trade off between degree of multiprogramming and throughput. The higher the number of processes, the smaller the time each of the processes may control CPU for. Long-term scheduler is also known as Admission Scheduler [21]. It is required to ensure that real time processes get enough CPU time to complete their tasks. The long term queue exists in the hard disk or virtual memory.

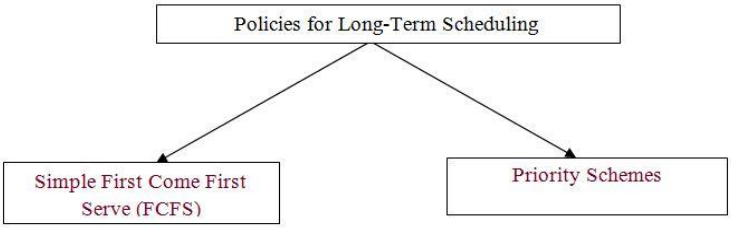

Figure 2.6: Long term scheduling policies

2.2.2 Midterm scheduling

The scheduling of processes is mainly done based on the requirement of the resources. It is essentially concerned with memory management and often designed as a memory management subsystem of an operating system. It temporarily removes a process from the main memory which is of low priority or has been inactive for a long time. This is known as "swamping out" of a process. The scheduler may decide to swamp out the process which is page-faulting frequently or a process which is taking large amount of memory [21]. Its efficient interaction with the short term scheduler is very essential for the performance of the systems with virtual memory.

2.2.3 Short term scheduling/CPU Scheduler

It is concerned with the allocation of CPU time to meet the processes in order to meet some pre-defined objectives. It decides which of the in-memory process is executed following an interrupt or operating system call. This scheduler makes more frequent scheduling decisions than long-term and midterm schedulers. [23,24] It can be pre-emptive or non-pre-emptive depending on the requirement. Mostly, it is written in assembler as it is a critical part of operating system.

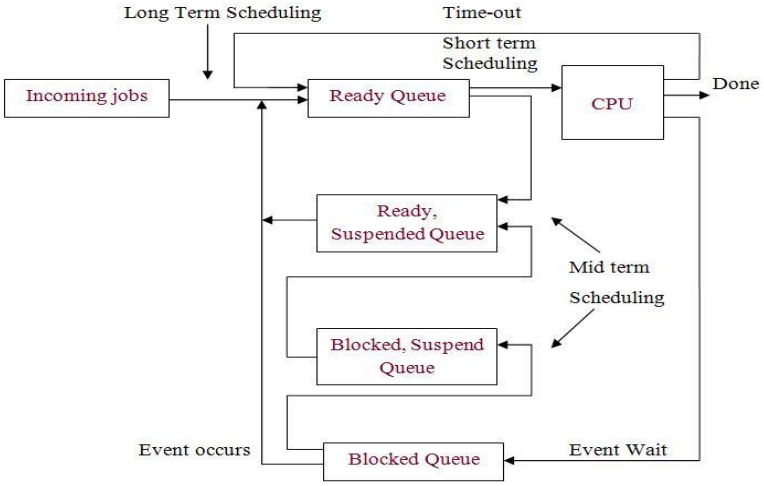

Figure 2.7: Queuing Diagram for scheduling

Chapter 3 Scheduling in Real Time Environment

In this chapter, the basics of real time systems are discussed. The concept of deadlines and their classification is given. Also the classification of real time systems on the basis of different parameters has been done. The various real time scheduling algorithms are discussed with the emphasis on Rate Monotonic and Earliest Deadline First algorithm. At last there is a comparison of various scheduling techniques to be used in real time environment.

3.1 Real Time Systems

The systems in which accuracy of the computation not only depend on the correctness of the results but also on the time at which the results are produced. In order to get accurate results, the real time systems should allow us to predict and control the occurrence of the task. The purpose of real time system in the physical world is to have a physical effect in a chosen time frame i.e. the accuracy of the result is time dependent.

Real time system consists of a controlling system which can be a real time computer and a controlled system which is the environment [19,25,31]. A real-time computer system is always part of a larger system – this larger system is called a real-time system. A real-time system changes as a function of physical time.

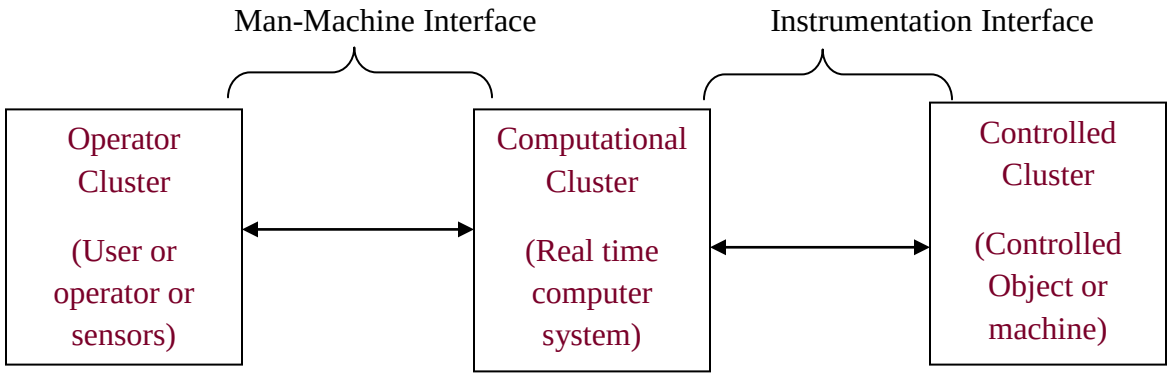

Figure 3.1: A Real Time System

A Real-time system can be decomposed into a set of self contained subsystems known as clusters, see figure 3.1. There can be broadly three clusters that can completely specify the real time system namely, operator cluster, computational cluster and controlled cluster. Operator cluster constitutes of operator or the user which use that system and the interface with the computational cluster is known as man-machine interface. Also, instead of user there can be

sensors and the computational cluster will respond to the information provided by those sensors. The computational cluster is responsible for the all the processing and computations of the system. The machine which is to be controlled constitute controlled cluster and the interface with the computational cluster is known as instrumentation interface.

Sensors will provide readings at regular intervals to the real time computer and the computer must respond by sending signals to actuators. The state of environment as received by the controlling system must be consistent with the actual state of environment for the proper response of the controlled system. Therefore, timely monitoring and processing of the sensed information is necessary.

There may be some irregular or unexpected events which must also receive a response and also there is an upper bound on the time (Deadline) within which the response should be delivered. The ability of the system to meet these demands depends on its capacity to perform number of computations in the given time [18]. If the events are occurring close enough then the computer needs to schedule those events so that each response is provided within the required time bounds. The system which is unable to fulfil all the unexpected demands lacks sufficient resources. A system with unlimited resources and capable of processing at infinite speed could satisfy any such timing constraint. Failing to meet the timing constraints can have different consequences and thus there is a classification of the deadlines:

- **Soft Deadline**: If the results produced after the deadline has passed and are still useful then this type of deadline is known as soft deadline. Reservation systems come under this category.
- **Firm deadline**: This deadline is one in which the results produced after the deadline is missed is of no utility. Infrequent deadline misses are tolerable. These types of deadlines are used in systems which are performing some important operations.
- **Hard deadline**: If catastrophe results on missing the deadline then this type of deadline is known as hard deadline. The systems which are performing critical applications like air traffic control come under this category.

3.2 Features of Real-Time Systems

A real-time system must have following features for the efficient working of the system:

- **Fast Response and Predictability**

  The systems should respond predictably fast to the urgent events so as their deadlines are met. The timely generation of results is essential for the accurate working of the system.

- **High degree of schedulability**

  The timing requirements must be satisfied at high degrees of resource usage [19]. The constraints should be met even if all the resources are being utilized i.e. high degree of schedulability should be achieved.

- **Stability under transient overload**

  The system must be stable under overloaded conditions which mean that the deadlines of the selected critical tasks must be met even if the system is overloaded and other deadlines could not be met. The system should be robust enough to deal with peak overload conditions.

- **Efficiency**

  As most real time systems are embedded into small devices with space, weight, memory, power constraints, an efficient management of available resources is essential for achieving a desired performance.

- **Fault Tolerance**

  The system should not crash on the occurrence of single hardware or software failure. So, the critical components have to be designed to be fault tolerant.

- **Maintainability**

  Architecture should be in modular form in order to perform easy modifications on the system and ensuring the enhancement of the system.

	Non Real-Time Systems	Real-Time Systems
Capacity	Capable of producing high throughput.	Must be capable of meeting all the deadlines i.e. high schedulability of the events.
Responsiveness	Aims at having fast average response	Aims at ensuring worst case response time to the events
Overload	These systems behave fairly under overload conditions.	These systems aim at getting stable in overload conditions. Deadlines of critical tasks are met even if other deadlines are missed.

Table 3.1: Comparison of real-time systems and non real-time systems

3.3 Real-Time Systems Application Domains

The areas where real-time systems find their applications are listed below:

- Telecommunication Systems
- Automotive Control
- Radar Systems
- Process Control
- Consumer Electronics
- Automated manufacturing systems
- Defense systems
- Semiconductor fabrication systems
- Air traffic control
- Autonomous navigation systems
- Vehicle control systems
- Satellite systems

3.4 Classification of Real Time Systems

Real-Time Systems can be classified into different categories on the basis of different perspectives. Broadly these are classified into two categories.

3.4.1 Classification on the basis of characteristics of application

This classification is dependent on the parameters which are not a part of the system rather depends on the factors outside the system. There can be further two classifications under this category.

3.4.1.1 Hard Real-Time Systems and Soft Real-Time Systems

The design of both the systems is different from each other. The characteristics of both these systems are fundamentally different.

- **Hard Real-Time Systems**

  The systems which require fast response time and missing a deadline may lead to catastrophe are known as hard real-time systems. These are highly autonomous so as to maintain fail safe operation of the process. Error detection and recovery of such systems are autonomous in order to quickly arrive at a safe state in case of failure. The key concern in hard real-time systems is on the short-term temporal accuracy of the real-time database that is invalidated by the flow of real-time. Rollback or recovery in these systems is of limited utility because guaranteeing the deadline after the occurrence of an error is difficult.

- **Soft Real-Time Systems**

  The response time of such systems is usually in seconds and missing a deadline would not result in a catastrophe. Degraded performance in the peak load conditions is tolerated in these systems. Error detection and recovery is non-critical in soft real-time systems. Long term integrity and availability of large data files is a key concern in such system and it should be maintained for the proper processing of the data. Computation can be rolled back in case of detection of error so as to initiate a recovery action.

3.4.1.2 Fail Safe and Fail Operational

This classification is based on the system's behavior in case of failure. The real-time systems have one or more safe states and the reaction of the system to a failure differentiates the systems.

- **Fail Safe Systems**

  The identification of safe state and approach to that state is possible upon the occurrence of failure in these systems. This characteristic to reach to the safe state is of the controlled device and not of the computer system controlling it. The computer systems of such real-time systems should have high error-detection coverage i.e. error detection probability should be near to one.

- **Fail Operational Systems**

  The systems in which identification of safe state is not possible and the system needs to remain operational on the occurrence of a failure are known as fail operational systems. The system must remain operational for sometime so as to avoid catastrophe on the occurrence of the failure. Flight control system aboard an airplane is an example of such system.

3.4.2 Classification on the basis of design of computer application

Real-Time systems can also be classified on the basis of the design of the computer application which depends on the factor inside the computer system. The real-time systems are divided into three categories under this classification

3.4.2.1 Guaranteed Response and Best Effort

The design of the system which considers the response of the system under peak load and fault scenario necessitates the classification of the systems.

- **Guaranteed Response**

  If a design is made which is adequate even at peak load and fault scenario then we say we have guaranteed response. The probability of failure of a perfect system with guaranteed response is reduced to the probability that the assumptions about the peak load and the number and types of faults do not hold in reality. This probability is called assumption coverage. Careful planning and extensive analysis is required during design phase.

- **Best effort**

  The design of such systems is done according to the principle of best possible effort taken and the sufficiency of the design is checked during the test and integration phases. Best-

effort systems do not require a rigorous specification of the load- and fault-hypothesis. Many non safety-critical real-time systems are designed according to the best-effort criteria.

3.4.2.2 Resource adequate and Resource inadequate

The availability of the resources to handle peak load and fault scenario becomes a basis of classification of real time system design.

- **Resource Adequate**

  In this system, there are enough computing resources available to handle the specified peak load and the fault scenario. Guaranteed response systems are based on the principle of resource adequacy. Hard real-time systems must be designed according to the guaranteed response paradigm that requires the availability of adequate resources.

- **Resource inadequate**

  This system is based on the theory that the provision of sufficient resources to handle every possible situation is not economically viable. So a dynamic resource allocation technique must be devised based on resource sharing. Many non safety-critical real-time system designs are based on the principle of resource inadequacy.

3.4.2.3 Event triggered and Time triggered

A trigger is an event which causes the start of some action in the computer. The distinction on the basis of trigger depends on the internal triggers and not the external behaviour of the real-time system.

- **Event Triggered**

  Any significant event other than regular event on clock initiates the processing activity. The significant event is signalled to the CPU by interrupt mechanism. These systems require dynamic scheduling strategy to activate appropriate process to service the event.

- **Time Triggered**

  All activities are initiated by the progression of real-time. There is only one interrupt in each node of a distributed time triggered system, the periodic real-time clock interrupt. Every communication or processing activity is initiated at a periodically occurring predetermined tick of a clock.

3.5 Schedulability Test

It is a test that determines whether the ready tasks can be scheduled in order to meet its specified deadlines. An optimal scheduler is one which can always find a feasible schedule whenever it exists [29]. The instant when the request of execution of task is made is known as task request time. There are two different task types based on the request times.

- **Periodic tasks**

  The period of these tasks is known and request for the such tasks occur after specified intervals of time. All the future request times are known beforehand by adding the multiples of known period to the initial request time.

- **Sporadic tasks**

  Sporadic tasks are real-time tasks which are activated irregularly with some known bounded rate. The bounded rate is characterized by a minimum interarrival period, that is, a minimum interval of time between two successive activations. This is necessary in order to bound the workload generated by such tasks. The time constraint is usually a deadline d.

  Scheduability test can be classified into three categories [19]

- **Sufficient schedulability test**

  The positive outcome of these tests guarantees that all the deadlines are always met.

  Sometimes the otcome of scheduable task can be negative.

- **Necessary schedulability test**

  The failure of this test will surely result in a deadline miss at some point of time. The task set is defininitely not schedulable if the outcome of this test is negative. A necessary schedulabilty test for a set of periodic tasks states that the sum of utilization factors must be less than or equal to total number of available processors.

- **Exact schedulability test**

  A test which is both sufficient and necessary is known as exact schedulability test. It has the features of sufficient schedulability test as well as necessary schedulability test.

3.6 Real-Time Software Architectures

The software architecture of a computing system is the structure(s) of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Most of the real-time architectures use one of the following five architectural patterns.

3.6.1 Cyclic Executives or Frame-Based Systems

A cyclic executive consists of continuously repeated task sequences, known as major frames.

Each major frame consists of minor frames which are small slices of time and tasks are scheduled into specific minor frames [32]. Triggering of the task in a frame is done using the timer. A set of minor cycles which are non-repeating constitute a major cycle. Every minor cycle contain a pre-defined list which contains operations implemented as procedures. Concurrency is not used and all the large operations need to be manually divided into small parts to fit the frame.

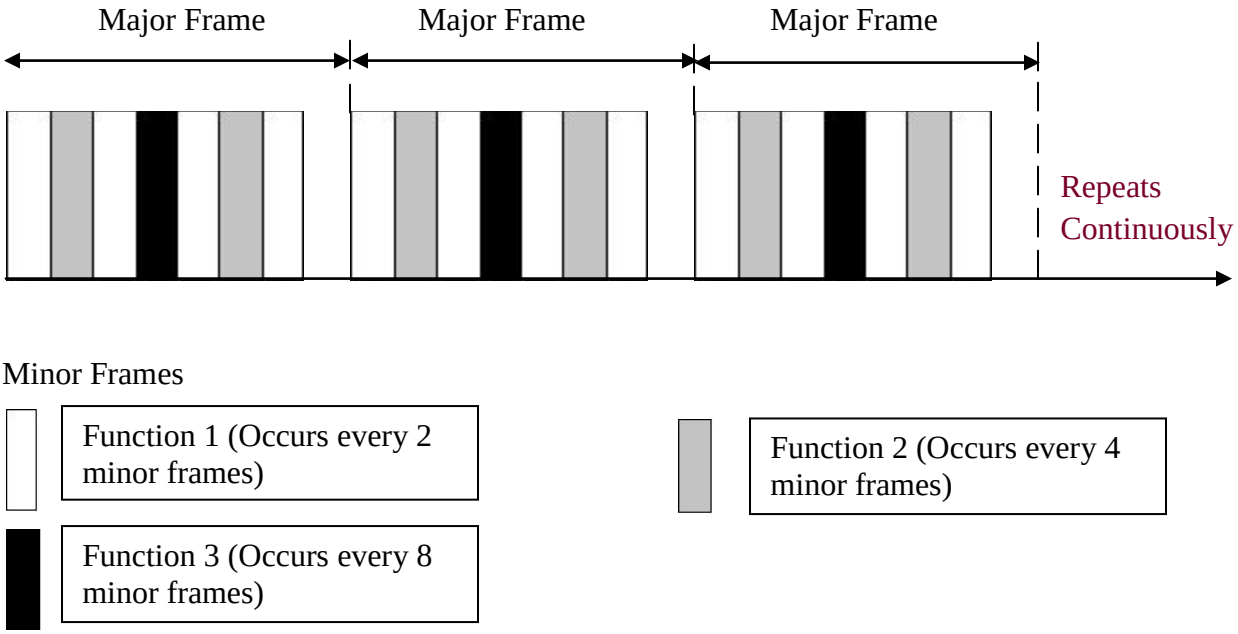

Figure 3.2: Cyclic Executives [32]

3.6.2 Event Driven Systems

Event Driven design schedules tasks using real time I/O completion events or timer events. Most of the commercial real time linux based systems uses this model. Priorities are determined by time constraints using rate monotonic or deadline monotonic priority assignment techniques. The scheduling of such systems is done at the runtime. Unbounded priority inversion must be avoided in order to get predictable response. Priorities should be fixed that can be determined on

the basis of the periods of the tasks or the deadlines. Utilization bounds specify the utilization of the resources by the tasks and this must be preserved during aperiodic events in order to preserve predictable response. These systems are pre-emptive in nature as the priority assignment schemes are inherently pre-emptive [32]. These systems are relatively simple and statically analyzable. They cannot work with dynamically changing load i.e. they work well with only fixed load conditions. These systems are not capable of handling distributed environments.

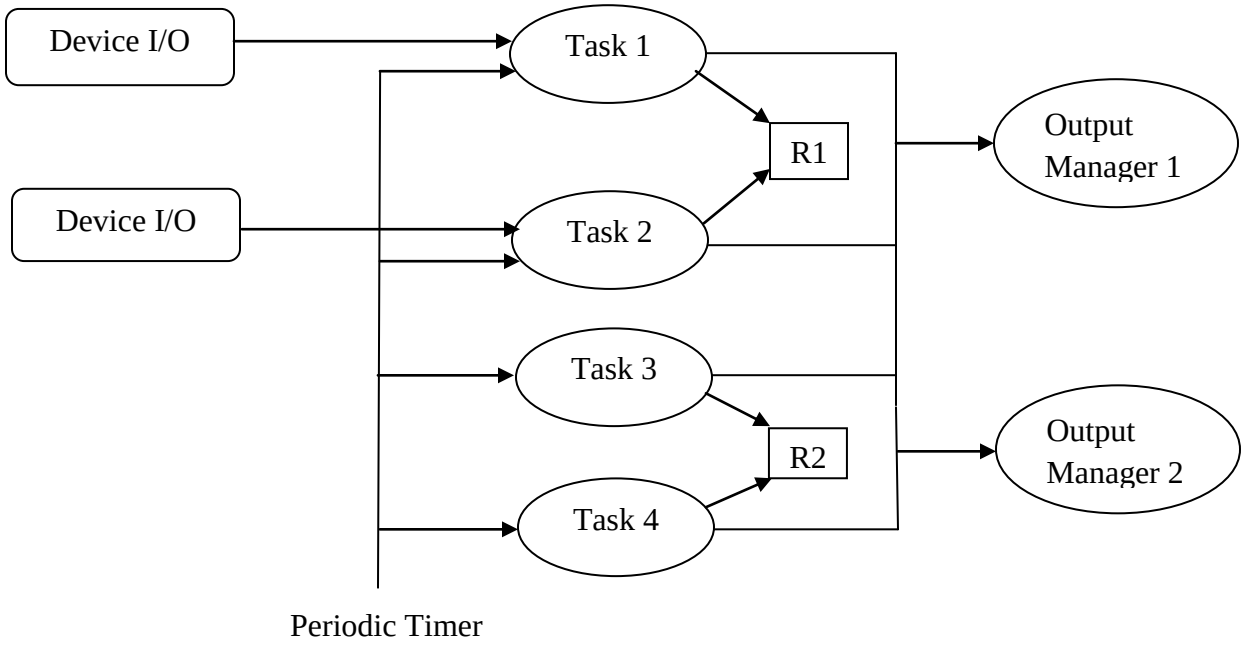

Figure 3.3: Event Driven Systems [32]

3.6.3 Pipelined Systems

As event driven systems uses I/O completion and periodic timers to trigger the scheduable tasks, pipelined systems use prioritized inter-task messages in addition to I/O completion and timers to trigger the tasks. Control flow for all the system proceeds from source to destination. Task priorities are set according to the direction of pipeline [32]. If pipeline is unidirectional, then message queue buildup will be minimum if set task priorities in increasing order and for the bi-directional pipeline, priorities are set in such a way that they are equal along the pipeline.

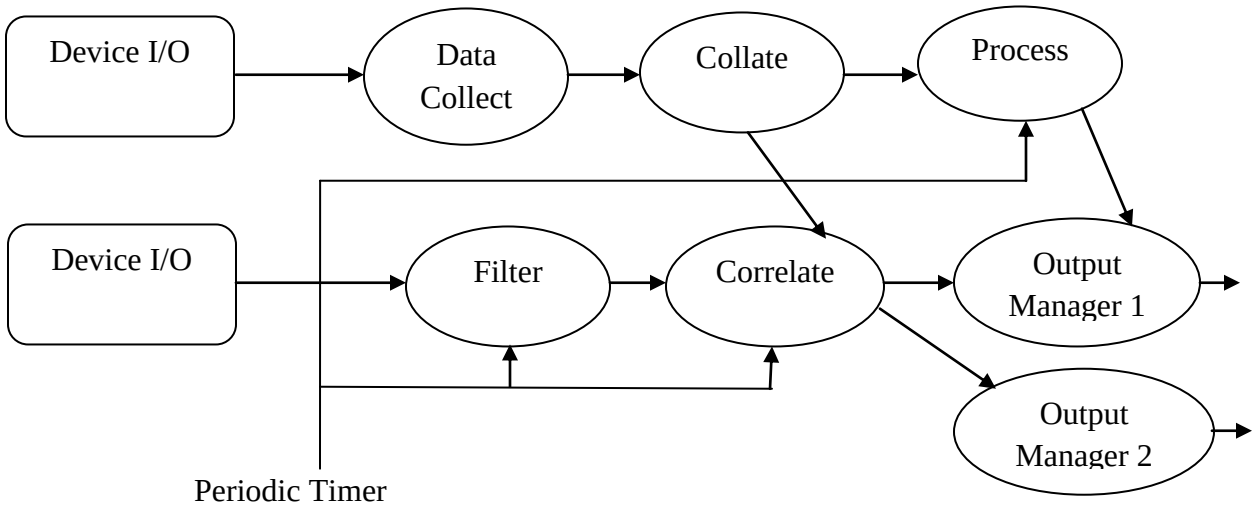

Figure 3.4: Pipelined Systems [32]

Analyzing a pipelined system is a difficult task because of their complexity. Rate Monotonic Analysis (RMA) techniques should be modified in order to account for message driven nature of pipelined systems as well as for precedence constraints. Also there is a challenge of sequencing the threads of pipelined tasks and they have to meet their timing constraints. So, pipelined real time systems are generally kept simple to ensure that all the deadines are met.

3.6.4 Client Server Systems

These systems are similar to pipelined systems in terms of analysis. They use inter-task messages in addition to periodic timer and I/O completion to trigger the schedulable tasks. Sending tasks (clients) are blocked until they get a response from receiving tasks (servers). Control of flow remain at a single system node while data flow is distrbuted. Error processing and debugging are significantly easier in such system in comparison to pipelined systems. Priorities in these systems play a minor role. Task priorities are generally same as server tasks inheriting priorities from clients is impractical.

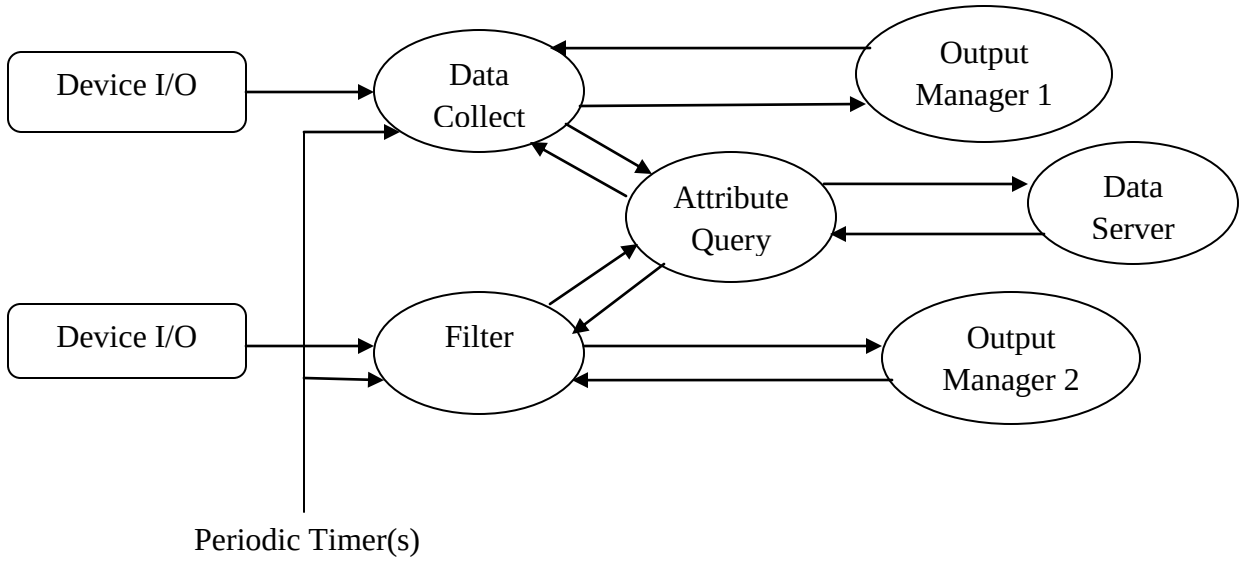

Figure 3.5: Client Server Systems [32]

3.6.5 State Machine Systems

In this architecture, the system is broken down into a set of concurrent extended finite state machines. Each finite state machine is typically used to model the behavior of a reactive or active object. At every moment, the object resides in one of a finite number of states, waiting for an event. A transition is triggered whenever there is an event which might change the state of the machine and some actions are executed associated with the transition. Pre-emption is generally not allowed in state machine systems [32]. A system with concurrent machines is implemented by mapping them to concurently executing tasks. Task priorities controls the timing analysis of state machine architecture. Ensuring that each task handles either single timing constraint or set of similar constraints can make the system analyzable by assigning priorities on the basis of timing constraints. This works well with Concurrent Object-Based Real-Time Analysis (C.O.B.R.A) systems but they are complex to analyze.

3.7 Dynamic Real-Time Scheduling Algorithms

The choice of the task which is to be serviced next is done at the run-time. The algorithms performing such scheduling differ in the assumptions about the complexity of task and task behaviour. Before any comprehensive theory was available, the most critical control applications were developed using timeline scheduling, according to which the time line is divided into slots of fixed length, called minor cycle and tasks are statically allocated in each slot based on their

rates and execution. The schedule is then constructed up to the major cycle which is the least common multiple of all the periods and stored in a table. Tasks are dispatched according to the schedule at the run-time which is synchronized by a timer at the beginning of each minor cycle. Timeline scheduline is easy to implement and does not introduce any runtime overhead but is fragile in overload conditions. Also, it is incapable of handling dynamic situations. So these problems can be solved using priority based approaches and schedule is generated at the runtime. These approaches are known as dynamic scheduling with either static priority assignment techniques or dynamic priority assignment techniques.

An **optimal real time scheduling algorithm** is one which may fail to meet any deadline only if no other algorithm can meet the deadline.

There are broadly four dynamic real-time scheduling algorithms which are commercially used and under research.

3.7.1 Rate Monotonic Scheduling (RMS) Algorithm

This is a static priority based dynamic scheduling algorithm given by Liu and Layland in 1973. Rate Monotonic Scheduling theory is used where finite number of periodic tasks share a single processor [28]. It decides whether the tasks can be scheduled according to their timing constraints without missing any deadline. It assigns priorities to tasks according to their request rates. The higher the request rate i.e. shorter the period, the higher is the priority. A priority P_i is assigned to a task and it remains same for whole time. RM scheduling is pre-emptive in nature i.e. executing task can be pre-empted by the arrival of the task with shorter period. A set of jobs is schedulable or feasible if all timing constraints are met, that is, all hard real-time jobs complete by their respective deadlines. An optimal real-time scheduling algorithm is one which may fail to meet a deadline only if no other scheduling algorithm can meet the deadline. It has been shown in 1973 that RMS algorithm is optimal among all fixed-priority assignments in the sense that no other fixed-priority algorithms can schedule a task set that cannot be scheduled by RM.

The assumptions made about the task set are mentioned below.

1. The request for all the task sets, for which hard deadlines should be met, are periodic.
2. All tasks are independent of each other. No precedence constraints or mutual exclusion constraints exist between any pair of tasks.
3. The deadline interval of every task is equal to its period.
4. The required maximum computation time is known beforehand and is constant.

5. Time required for context switching can be ignored.
6. Sum of utilization factors of n tasks with period period p is given by

$$U = \sum \frac{C_i}{P_i} \leq n\left(2^{1/n} - 1\right)$$

In order to understand RM scheduling technique [19], consider two tasks τ_1 and τ_2 with computation times as C_1 and C_2 respectively and periods T_1 and T_2 respectively. Consider $T_1 < T_2$, so according to the RM scheduling algorithm, τ_1 will get higher priority. For guaranteeing feasible schedule, we take two cases.

Case 1: C_1 is short enough that all its requests gets completed before the second request of τ_2. Let $F = \lfloor \frac{T_2}{T_1} \rfloor$ be the number of periods of τ_1 entirely contained in T_2. Here,

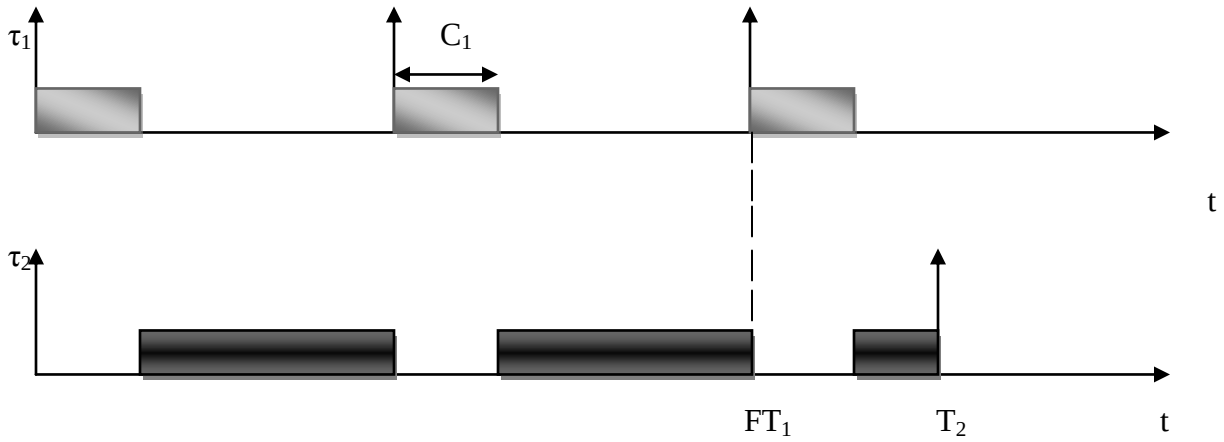

Figure 3.6: Scheduling tasks τ_1 and τ_2 using RM schedule with condition, $C_1 < T_2 - FT_1$

$$C_1 < T_2 - FT_1 \quad \ldots(3.1)$$

From the above waveform, we can see that the task set is schedulable if

$$(F+1)C_1 + C_2 \leq T_2 \quad \ldots(3.2)$$

Also the task is schedulable by any other algorithm if, $C_1 + C_2 \leq T_1$ …(3.3)

Multiplying both sides by F, we get

$$FC_1 + FC_2 \leq FT_1 \quad \ldots(3.4)$$

and we know in this case $F \geq 1$, we can write

$$FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1 \quad \ldots(3.5)$$

Adding C_1 to each term, we get

$$(F+1)C_1 + C_2 \leq (F+1)C_1 + FC_2 \leq FT_1 + C_1 \quad \ldots(3.6)$$

As we have assumed that, $C_1 < T_2 - FT_1$ (3.1) which gives, $FT_1 + C_1 < T_2$

Substituting this in equation (3.6), we get

$$(F+1)C_1 + C_2 \leq FT_1 + C_1 < T_2 \quad \ldots(3.7)$$

Comparing equation (3.2) and (3.7), we see that both are same and hence the task is schedulable if equation (3.2) stands true for this particular case.

Case 2. C_1 is long enough that it overlap with the second request of τ_2. This condition can be depicted by $C_1 \geq T_2 - FT_1$

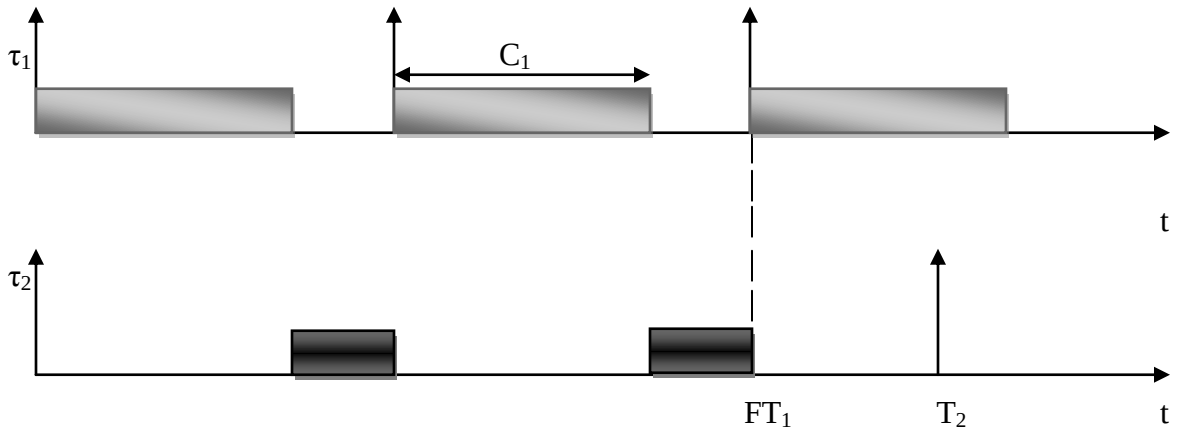

Figure 3.7: Scheduling tasks τ_1 and τ_2 using RM schedule with condition, $C_1 \geq T_2 - FT_1$

$$\boldsymbol{C_1 \geq T_2 - FT_1} \quad \ldots(3.8)$$

From the above figure it is clear that the tasks τ_1 and τ_2 with periods T_1 and T_2 ($T_2 > T_1$) respectively are schedulable if following condition is satisfied,

$$FC_1 + C_2 \leq FT_1 \quad \ldots(3.9)$$

Again we start from the same equation that for two tasks with periods T_1 and T_2 to be schedulable under any algorithm should satisfy the equation (3.3), which is,

$C_1 + C_2 \leq T_1$

Multiplying it by F, we get

$FC_1 + FC_2 \leq FT_1$ …(3.10)

and we know $F \geq 1$, we can write

$FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1$ …(3.11)

So we have equation (3.11) which is same as equation (3.9). So if the tasks can be scheduled using any priority assignment method then it can be scheduled using RM also [19]. Hence RM is optimal priority scheduling algorithm.

3.7.2 Deadline Monotonic (DM)

This technique is an extension of Rate Monotonic scheduling algorithm. This is first proposed in 1982 by Leung and Whiteland. This is also fully preemptive technique used for scheduling tasks with static priorities [28]. The third assumption mentioned in rate monotonic technique that says the deadline interval of every task is same and equal to its period; has been relaxed. The tasks have constrained deadlines i.e. relative deadlines can be less than or equal to its period.

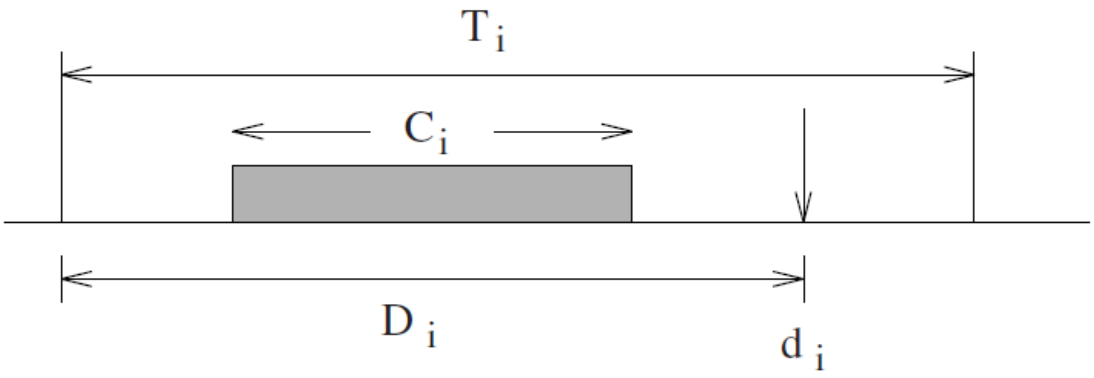

Figure 3.8: Task parameters in Deadline Monotonic scheduling [19]

C_i: Worst case computation time

D_i: Relative Deadline

T_i: Period of the task

Each task is assigned a fixed priority inversely proportional to its relative deadline D_i. As relative deadlines are constant, DM is a static priority assignment technique.

Deadline monotonic priority assignment algorithm is optimal which means that if the task is schedulable by any of the static priority assignment scheme then it is also schedulable using DM.

3.7.3 Earliest Deadline First (EDF)

It is the optimal dynamic preemptive algorithm for a single processor systems which are based on dynamic priorities. The tasks with the earliest deadline is given the highest priority. That is why this algorithm is also known as deadline driven scheduling algorithm(DDS). The jobs in the task set are put in the ready queue on the basis of their priorities. So, this algorithm is a job level fixed-priority algorithm. A task set can be scheduled by this algorithm if the utilization factor, $U \leq 1$. It is an optimal run-time scheduler which executes the ready task having earliest absolute deadline (d_i) first at every instant. Absolute deadline,d_i, is given by,

$d_i = r_i + D_i$

Where, r_i is the arrival time of the task and D_i is the relative deadline of the task.

If more than one task have the same deadline, EDF randomly selects one for execution next. EDF is a dynamic-priority scheduler since task priorities may change at run-time depending on the nearness of their absolute deadlines [30].

To explain the optimality of the Earliest Deadline First (EDF), consider an example. If we have 4 tasks with the following information,

$$J_1: r_1 = 0;\ C_1 = 3;\ D_1 = 16$$

$$J_2: r_2 = 0;\ C_2 = 5;\ D_2 = 14$$

$$J_3: r_3 = 3;\ C_3 = 6;\ D_3 = 9$$

$$J_4: r_4 = 4;\ C_4 = 2;\ D_4 = 10$$

So, if we have to schedule these 4 tasks on the processor using FIFO (First In First Out) technique or FCFS (First Come First Serve) technique, the schedule would be like

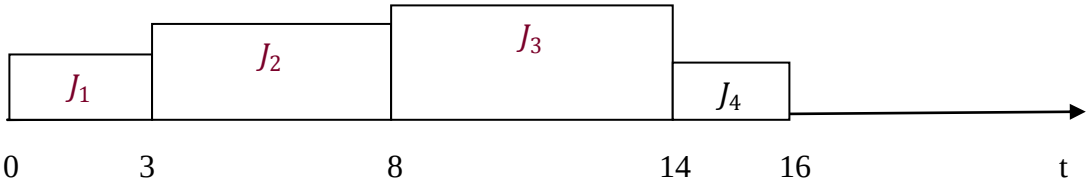

Figure 3.9: FIFO Schedule

From the above schedule, it is clear that tasks were scheduled on the basis of their arrival time and the tasks which arrive first were given higher priority regardless of their deadlines. It can be seen that the task J_3 which arrives at 3rd interval and has computation time of 6 unit time, misses its deadline. Also, the task J_4 misses its deadline which is 12 unit time (relative deadline).

If we schedule the same task set using EDF schedule, the task sets will be scheduled on the basis of their absolute deadlines and the following schedule is observed.

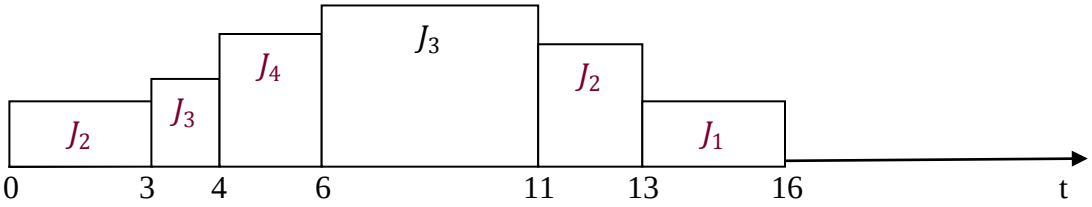

Figure 3.10: EDF Schedule

From the EDF Schedule it is clear that the tasks are scheduled on the basis of their absolute deadlines and the incoming tasks with closer deadline pre-empt the executing task to start its execution. Also, the schedule given by EDF technique is optimal and all the tasks have completed their execution within their deadline.

Given a set *S* of independent and pre-emptable asks with arbitrary start times and deadlines on a uniprocessor, the EDF algorithm yields a feasible schedule for *S* if and only if *S* has feasible schedules [30]. Therefore, the EDF algorithm fails to meet a deadline of a task set satisfying the above constraints only if no other scheduler can produce a feasible schedule for this task set. The proof of EDF's optimality is based on the fact that any non-EDF schedule can be transformed into an EDF schedule.

Schedulability Test

Let C_i denote the computation time of task J_i. For a set of N periodic tasks such that the relative deadline D_i of each task is equal to or greater than its respective period T_i $(D_i \geq T_i)$, a necessary and sufficient condition for feasible scheduling of this task set on a uniprocessor is that the utilization of the tasks is less than or equal to 1:

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

A sufficient condition for feasible scheduling [30] of a set of independent, pre-emptable, and periodic tasks on a uniprocessor is,

$$\sum_{i=1}^{N} \frac{C_i}{min(D_i, T_i)} \leq 1$$

3.7.4 Least Laxity algorithm (LL)

It is an optimal algorithm in single processor systems and is also known as Minimum Laxity First(MLF) or Least Slack First(LST). Laxity of the task is defined as a difference between deadline interval and maximun computation. Laxity of a task i,

$$l_i = D_i - C_i$$

D_i is the relative deadline interval of task i, which is the duration between the deadline of the task and the task request instant.

C_i is the computation time of task i.

Thus, the laxity of the task is maximum time the task can delay execution without missing its deadline in the future. The assumptions about the task sets are same as that of EDF algorithm. The priorities are assigned on the basis of laxity. The task with longest laxity gets the lowest priority[19,30]. This may cause frequent switching between the tasks and thus increase the system overhead.

Different priorities are assigned to different instances of the same task, thus the priority of each task may change with respect to other tasks as new task instances arrive and complete.

For the task set with relative deadline D_i, computation time C_i and period T_i, utilization of the task should be less than or equal to 1 for the feasible schedule.

$$\sum_{i=1}^{N} \frac{C_i}{min(D_i, T_i)} \leq 1$$

3.8 Comparison of Real Time Scheduling Algorithms

	Rate Monotonic	Deadline Monotonic	Earliest Deadline First	Least Laxity First
Implementation	Simplest	Simple	Difficult	Difficult
Processor Utilization	Less	More as compared to RM	Full Utilization	Full Utilization
Pre-emption	Allowed and fixed for a particular task set	Fixed for a particular task set	Allowed but not fixed and generally high than static priority techniques	More number of pre-emptions as compared to EDF
Priority Assignment	Static	Static	Dynamic	Dynamic
Scheduling criterion	Task Period	Relative Deadline	Deadline	Laxity i.e. d_i-c_i
Jitter Control	Only for highest priority task	Only for highest priority task	Inefficient in overloaded systems	Inefficient in overloaded systems

Table 3.2: Comparison of Real Time Scheduling Algorithms

Chapter 4 Results and Discussions

From the literature survey, it has been observed that there is a need to find shortest paths in a communication network for scheduling tasks in the network. Shortest path helps in optimizing common resources in the network and efficient schedule is obtained. The shortest paths and the probability of error has been evaluated and simulated results has been achieved by using TORA software.

There are some variations in the shortest path problems and work has been carried out by dividing them in three categories.

Single source shortest path problem

If a single source is used to transmit data to all other nodes then the problem is known as single source shortest path problem. This problem deals with finding the shortest path from a single source to all the nodes or vertices in the graph.

Single destination shortest path problem

The shortest paths from all the vertices in the directed graph to a single vertex v has to be found. This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.

All-pairs shortest path problem

Shortest path between every pair of vertices v and v' has to be calculated and thus the name All-pairs shortest path problem.

The tasks need to be scheduled through the network shown in figure 4.1 and the optimized shortest paths have to be calculated so as to achieve optimized resource utilization. The network shown in figure 4.1 is the symmetrical undirected graph with values written on the edges signifies delay between the nodes in ns. The number of intermediate nodes and the delay along the different paths needs to be evaluated which is shown in subsequent sections.

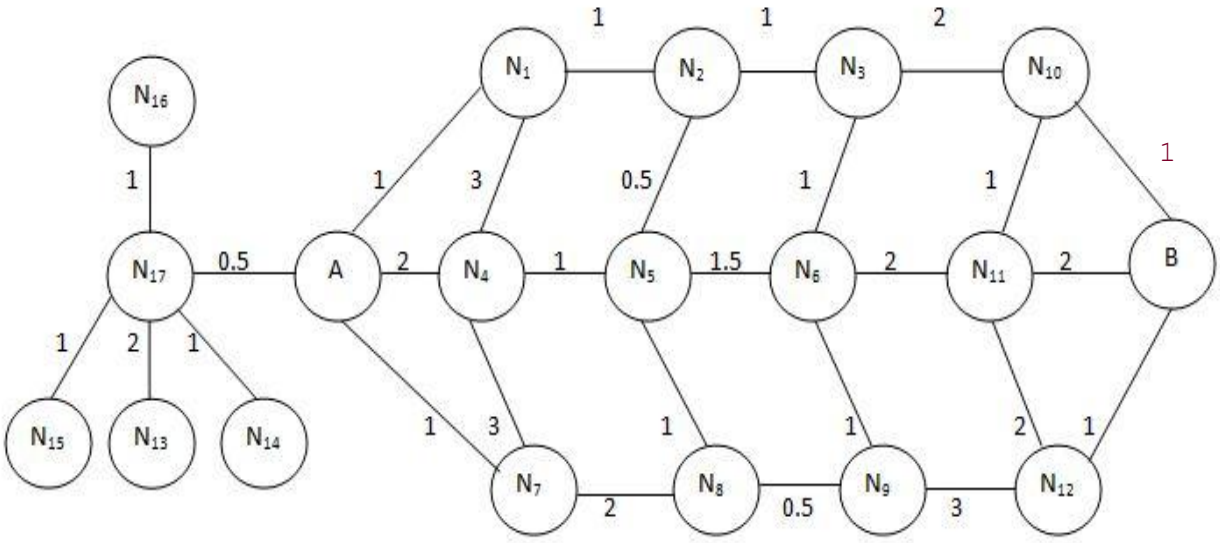

Figure 4.1: Undirected graph of the Problem

The network has been critically analyzed and the shortest path for node $N5$ to all other nodes has been achieved. The table 4.1 shows the name of the node, the various paths to that node from node $N5$, the delay along the path and the number of intermediate nodes.

S. No.	Node	Paths	Delay(ns)	Number of Intermediate Nodes
1.	$N1$	$N5 \xrightarrow{1} N4 \xrightarrow{3} N1$	4	1
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N1$	1.5	1
		$N5 \xrightarrow{1} N8 \xrightarrow{2} N7 \xrightarrow{1} A \xrightarrow{1} N1$	9	3
		$N5 \xrightarrow{1} N4 \xrightarrow{2} A \xrightarrow{1} N1$	4	2
2.	$N2$	$N5 \xrightarrow{0.5} N2$	0.5	0
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N1 \xrightarrow{1} N2$	5	2
		$N5 \xrightarrow{1.5} N6 \xrightarrow{1} N3 \xrightarrow{1} N2$	3.5	2
3.	$N3$	$N5 \xrightarrow{1.5} N6 \xrightarrow{1} N3$	2.5	1
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N3$	1.5	1
		$N5 \xrightarrow{1} N8 \xrightarrow{0.5} N9 \xrightarrow{1} N6 \xrightarrow{1} N3$	3.5	3

		$N5 \overset{1}{\rightarrow} N4 \overset{3}{\rightarrow} N1 \overset{1}{\rightarrow} N2 \overset{1}{\rightarrow} N3$	6	3
		$N5 \overset{1.5}{\rightarrow} N6 \overset{2}{\rightarrow} N11 \overset{1}{\rightarrow} N10 \overset{2}{\rightarrow} N3$	6.5	3
4.	$N4$	$N5 \overset{1}{\rightarrow} N4$	1	0
		$N5 \overset{0.5}{\rightarrow} N2 \overset{1}{\rightarrow} N1 \overset{3}{\rightarrow} N4$	4.5	2
		$N5 \overset{1}{\rightarrow} N8 \overset{2}{\rightarrow} N7 \overset{3}{\rightarrow} N4$	6	2
5.	$N5$	-	0	-
6.	$N6$	$N5 \overset{1.5}{\rightarrow} N6$	1.5	0
		$N5 \overset{0.5}{\rightarrow} N2 \overset{1}{\rightarrow} N3 \overset{1}{\rightarrow} N6$	2.5	2
		$N5 \overset{1}{\rightarrow} N8 \overset{0.5}{\rightarrow} N9 \overset{1}{\rightarrow} N6$	2.5	2
7.	$N7$	$N5 \overset{1}{\rightarrow} N8 \overset{2}{\rightarrow} N7$	3	1
		$N5 \overset{1}{\rightarrow} N4 \overset{3}{\rightarrow} N7$	4	1
		$N5 \overset{0.5}{\rightarrow} N2 \overset{1}{\rightarrow} N1 \overset{3}{\rightarrow} N4 \overset{3}{\rightarrow} N7$	7.5	3
		$N5 \overset{0.5}{\rightarrow} N2 \overset{1}{\rightarrow} N1 \overset{1}{\rightarrow} A \overset{1}{\rightarrow} N7$	3.5	3
		$N5 \overset{1.5}{\rightarrow} N6 \overset{1}{\rightarrow} N9 \overset{0.5}{\rightarrow} N8 \overset{2}{\rightarrow} N7$	5	3
		$N5 \overset{1}{\rightarrow} N4 \overset{2}{\rightarrow} A \overset{1}{\rightarrow} N7$	4	2
8.	$N8$	$N5 \overset{1}{\rightarrow} N8$	1	0
		$N5 \overset{1}{\rightarrow} N4 \overset{3}{\rightarrow} N7 \overset{2}{\rightarrow} N8$	6	2
		$N5 \overset{1.5}{\rightarrow} N6 \overset{1}{\rightarrow} N9 \overset{0.5}{\rightarrow} N8$	3	2
9.	$N9$	$N5 \overset{1.5}{\rightarrow} N6 \overset{1}{\rightarrow} N9$	2.5	1
		$N5 \overset{1}{\rightarrow} N8 \overset{0.5}{\rightarrow} N9$	1.5	1
		$N5 \overset{0.5}{\rightarrow} N2 \overset{1}{\rightarrow} N3 \overset{1}{\rightarrow} N6 \overset{1}{\rightarrow} N9$	3.5	3
		$N5 \overset{1}{\rightarrow} N4 \overset{3}{\rightarrow} N7 \overset{2}{\rightarrow} N8 \overset{0.5}{\rightarrow} N9$	6.5	3
10.	$N10$	$N5 \overset{1.5}{\rightarrow} N6 \overset{1}{\rightarrow} N3 \overset{2}{\rightarrow} N10$	4.5	2

		$N5 \xrightarrow{1.5} N6 \xrightarrow{2} N11 \xrightarrow{1} N10$	4.5	2
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N3 \xrightarrow{2} N10$	3.5	2
		$N5 \xrightarrow{1.5} N6 \xrightarrow{2} N11 \xrightarrow{2} B \xrightarrow{1} N10$	6.5	3
		$N5 \xrightarrow{1.5} N6 \xrightarrow{1} N9 \xrightarrow{3} N12 \xrightarrow{1} B \xrightarrow{1} N10$	8.5	4
11.	$N11$	$N5 \xrightarrow{1.5} N6 \xrightarrow{2} N11$	3.5	1
		$N5 \xrightarrow{1} N8 \xrightarrow{0.5} N9 \xrightarrow{3} N12 \xrightarrow{2} N11$	6.5	3
		$N5 \xrightarrow{1.5} N6 \xrightarrow{1} N3 \xrightarrow{2} N10 \xrightarrow{1} N11$	5.5	3
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N3 \xrightarrow{2} N10 \xrightarrow{1} N11$	4.5	3
		$N5 \xrightarrow{1.5} N6 \xrightarrow{1} N9 \xrightarrow{3} N12 \xrightarrow{2} N11$	7.5	3
12.	$N12$	$N5 \xrightarrow{1.5} N6 \xrightarrow{2} N11 \xrightarrow{2} N12$	5.5	2
		$N5 \xrightarrow{1} N8 \xrightarrow{0.5} N9 \xrightarrow{3} N12$	4.5	2
		$N5 \xrightarrow{1.5} N6 \xrightarrow{2} N11 \xrightarrow{2} B \xrightarrow{1} N12$	6.5	3
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N3 \xrightarrow{2} N10 \xrightarrow{1} B \xrightarrow{1} N12$	5.5	4
13.	$N13$	$N5 \xrightarrow{1} N4 \xrightarrow{2} A \xrightarrow{0.5} N17 \xrightarrow{2} N13$	5.5	3
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N1 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{2} N13$	5	4
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N1 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{2} N13$	7.5	4

		$N5 \xrightarrow{1} N8 \xrightarrow{2} N7 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{2} N13$	6.5	4
14.	$N14$	$N5 \xrightarrow{1} N4 \xrightarrow{2} A \xrightarrow{0.5} N17 \xrightarrow{1} N14$	4.5	3
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N1 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N14$	4	4
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N1 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N14$	6.5	4
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N7 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N14$	6.5	4
15.	$N15$	$N5 \xrightarrow{1} N4 \xrightarrow{2} A \xrightarrow{0.5} N17 \xrightarrow{1} N15$	4.5	3
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N1 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N15$	4	4
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N1 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N15$	6.5	4
		$N5 \xrightarrow{1} N8 \xrightarrow{2} N7 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N15$	5.5	4
16.	$N16$	$N5 \xrightarrow{1} N4 \xrightarrow{2} A \xrightarrow{0.5} N17 \xrightarrow{1} N16$	4.5	3
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N1 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N16$	4	4
		$N5 \xrightarrow{1} N8 \xrightarrow{2} N7 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N16$	5.5	4
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N7 \xrightarrow{1} A \xrightarrow{0.5} N17 \xrightarrow{1} N16$	6.5	4
17.	$N17$	$N5 \xrightarrow{1} N4 \xrightarrow{2} A \xrightarrow{0.5} N17$	3.5	2

		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N1 \xrightarrow{1} A \xrightarrow{0.5} N17$	3	3
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N1 \xrightarrow{1} A \xrightarrow{0.5} N17$	5.5	3
		$N5 \xrightarrow{1} N8 \xrightarrow{2} N7 \xrightarrow{1} A \xrightarrow{0.5} N17$	4.5	3
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N7 \xrightarrow{1} A \xrightarrow{0.5} N17$	5.5	3
18.	A	$N5 \xrightarrow{1} N4 \xrightarrow{2} A$	3	1
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N1 \xrightarrow{1} A$	2.5	2
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N1 \xrightarrow{1} A$	5	2
		$N5 \xrightarrow{1} N8 \xrightarrow{2} N7 \xrightarrow{1} A$	4	2
		$N5 \xrightarrow{1} N4 \xrightarrow{3} N7 \xrightarrow{1} A$	5	2
		$N5 \xrightarrow{1} N8 \xrightarrow{2} N7 \xrightarrow{3} N4 \xrightarrow{2} A$	8	3
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N1 \xrightarrow{3} N4 \xrightarrow{2} A$	6.5	3
19.	B	$N5 \xrightarrow{1.5} N6 \xrightarrow{2} N11 \xrightarrow{2} B$	5.5	2
		$N5 \xrightarrow{1.5} N6 \xrightarrow{1} N3 \xrightarrow{2} N10 \xrightarrow{1} B$	5.5	3
		$N5 \xrightarrow{1.5} N6 \xrightarrow{1} N9 \xrightarrow{3} N12 \xrightarrow{1} B$	6.5	3
		$N5 \xrightarrow{1.5} N6 \xrightarrow{2} N11 \xrightarrow{1} N10 \xrightarrow{1} B$	5.5	3
		$N5 \xrightarrow{1.5} N6 \xrightarrow{2} N11 \xrightarrow{2} N12 \xrightarrow{1} B$	6.5	3
		$N5 \xrightarrow{0.5} N2 \xrightarrow{1} N3 \xrightarrow{2} N10 \xrightarrow{1} B$	4.5	3
		$N5 \xrightarrow{1} N8 \xrightarrow{0.5} N9 \xrightarrow{3} N12 \xrightarrow{1} B$	5.5	3

Table 4.1: Various paths from node $N5$ to all other nodes

Similarly shortest routes from remaining nodes has been found. By knowing the number of intermediate nodes and the shortest paths between all the nodes in the network, the path between any two nodes has been optimized. The subsequent sections shows the optimized paths from every node to all other nodes and also contains the calculation of probability of error in case of failure of any one of the intermediate node.

For node $N1$, there are various paths to the other nodes and many nodes has to be traversed for reaching that node. For the transmission of the data to a particular node from $N1$ there can be multiple paths but we need that path in which there is minimum delay or the intermediate nodes are minimum.

From	To	Distance	Route
1-N1	2-N2	1.0	1-2
1-N1	3-N3	2.0	1-2-3
1-N1	4-N4	2.5	1-2-5-4
1-N1	5-N5	1.5	1-2-5
1-N1	6-N6	3.0	1-2-3-6
1-N1	7-N7	2.0	1-18-7
1-N1	8-N8	2.5	1-2-5-8
1-N1	9-N9	3.0	1-2-5-8-9
1-N1	10-N10	4.0	1-2-3-10
1-N1	11-N11	5.0	1-2-3-6-11
1-N1	12-N12	6.0	1-2-5-8-9-12
1-N1	13-N13	3.5	1-18-17-13
1-N1	14-N14	2.5	1-18-17-14
1-N1	15-N15	2.5	1-18-17-15
1-N1	16-N16	2.5	1-18-17-16
1-N1	17-N17	1.5	1-18-17
1-N1	18-A	1.0	1-18
1-N1	19-B	5.0	1-2-3-10-19

Figure 4.2: Shortest paths to all other nodes from node $N1$

From the results obtained above it has been observed that transmitting the data from node $N1$ to node $N12$ takes maximum time i.e. $6\,ns$ and has 4 intermediate nodes. If another path has been taken from $N1$ to node $N12$ then time taken has been found greater than or equal to $6\,ns$. If we take path $N1 \rightarrow N2 \rightarrow N5 \rightarrow N8 \rightarrow N9 \rightarrow N12$, it will take $7\,ns$ and number of intermediate nodes will be 4 only. Hence there will not be any change in the terms of intermediate nodes but the time taken has been reduced by,

$\frac{1}{7} \times 100 = 14.28\%$

So shortest paths to all the nodes has been found which are helpful in resolving timing constraints and saving of resources.

If any of the 4 intermediate nodes fails then the schedule will eventually fail. If we consider equal probability of failure of all nodes to be $\frac{1}{10}$, the probability of failure in optimized path can be given by, $\sum_{r=1}^{n} {}_{r}^{n}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$, which in this case comes out to be 0.34.

The following result is for the single source destination problem with $N2$ be the source. All the shortest paths has been found for all other nodes.

From	To	Distance	Route
2-N2	1-N1	1.0	2-1
2-N2	3-N3	1.0	2-3
2-N2	4-N4	1.5	2-5-4
2-N2	5-N5	0.5	2-5
2-N2	6-N6	2.0	2-3-6
2-N2	7-N7	3.0	2-1-18-7
2-N2	8-N8	1.5	2-5-8
2-N2	9-N9	2.0	2-5-8-9
2-N2	10-N10	3.0	2-3-10
2-N2	11-N11	4.0	2-3-6-11
2-N2	12-N12	5.0	2-5-8-9-12
2-N2	13-N13	4.5	2-1-18-17-13
2-N2	14-N14	3.5	2-1-18-17-14
2-N2	15-N15	3.5	2-1-18-17-15
2-N2	16-N16	3.5	2-1-18-17-16
2-N2	17-N17	2.5	2-1-18-17
2-N2	18-A	2.0	2-1-18
2-N2	19-B	4.0	2-3-10-19

Figure 4.3: Shortest paths from node $N2$ to all other nodes

It has been observed that it takes maximum amount of time to reach node $N12$ from node $N2$, which is $5\,ns$ and 3 intermediate nodes are transversed. If we choose any other path like $N2 \rightarrow N5 \rightarrow N10 \rightarrow N11 \rightarrow N12$, then data will take $6\,ns$ to reach $N12$ from node $N2$ and intermediate nodes will remain 3 only. Hence, the time needed is reduced by,

$\frac{1}{6} \times 100 = 16.67\%$

In this case, there is no change in the number of nodes traversed in order to reach node $N12$ from $N2$ but the time required to reach that node is reduced by 16.67%.

The probability of failure of the schedule in the optimized path which 3 intermediate nodes from node $N2$ to node $N12$, is given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

In this case $n = 3$, so the probability of failure of the schedule in the worst case comes out to be 0.27 approximately and this is lesser than the probability of failure on the route from node $N1$ node $N12$, as observed in previous case.

The node $N3$ connects all other nodes through the chain of intermediate nodes. The following result describes the shortest path from node $N3$ to all other nodes.

From	To	Distance	Route
3-N3	1-N1	2.0	3-2-1
3-N3	2-N2	1.0	3-2
3-N3	4-N4	2.5	3-2-5-4
3-N3	5-N5	1.5	3-2-5
3-N3	6-N6	1.0	3-6
3-N3	7-N7	4.0	3-2-1-18-7
3-N3	8-N8	2.5	3-2-5-8
3-N3	9-N9	2.0	3-6-9
3-N3	10-N10	2.0	3-10
3-N3	11-N11	3.0	3-6-11
3-N3	12-N12	4.0	3-10-19-12
3-N3	13-N13	5.5	3-2-1-18-17-13
3-N3	14-N14	4.5	3-2-1-18-17-14
3-N3	15-N15	4.5	3-2-1-18-17-15
3-N3	16-N16	4.5	3-2-1-18-17-16
3-N3	17-N17	3.5	3-2-1-18-17
3-N3	18-A	3.0	3-2-1-18
3-N3	19-B	3.0	3-10-19

Figure 4.4: Shortest paths from node $N3$ to all other nodes

It has been observed that it takes maximum time to reach node $N13$ from node $N3$ and time is $5.5\,ns$. There are 4 intermediate nodes which has to be crossed to reach node $N13$ from node $N3$. If any other path is taken to node $N13$ from node $N3$ which is $N3 \rightarrow N2 \rightarrow N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, the time needed to reach node $N13$ from $N3$ is $7\,ns$ and intermediate nodes to be traversed has been increased to 5. The time needed has been reduced by,

$$\frac{1.5}{7} \times 100 = 21.43\%$$

The number of intermediate nodes has been reduced by 1 along with 21.43% reduction in time.

The optimized path from node $N3$ to node $N13$ has 4 intermediate nodes. In order to ensure feasible schedule all nodes should function properly, the probability of failure in this case is given by,

$$\sum_{r=1}^{n} {}_{r}^{n}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

For $n = 4$, the value comes out to be 0.34 which is same as observed in the path from $N1$ to $N12$.

The following result gives the shortest path to all the nodes from node $N4$.

From	To	Distance	Route
4-N4	1-N1	2.5	4-5-2-1
4-N4	2-N2	1.5	4-5-2
4-N4	3-N3	2.5	4-5-2-3
4-N4	5-N5	1.0	4-5
4-N4	6-N6	2.5	4-5-6
4-N4	7-N7	3.0	4-7
4-N4	8-N8	2.0	4-5-8
4-N4	9-N9	2.5	4-5-8-9
4-N4	10-N10	4.5	4-5-2-3-10
4-N4	11-N11	4.5	4-5-6-11
4-N4	12-N12	5.5	4-5-8-9-12
4-N4	13-N13	4.5	4-18-17-13
4-N4	14-N14	3.5	4-18-17-14
4-N4	15-N15	3.5	4-18-17-15
4-N4	16-N16	3.5	4-18-17-16
4-N4	17-N17	2.5	4-18-17
4-N4	18-A	2.0	4-18
4-N4	19-B	5.5	4-5-2-3-10-19

Figure 4.5: Shortest paths from node $N4$ to all other nodes

From the result, it has been observed that it takes maximum time to reach node $N12$ and node B from node $N4$ which is $5.5\,ns$. The intermediate nodes which has to be crossed for node $N12$ and node B were 3 and 4 respectively . If any other path has been taken from $N4$, $N4 \rightarrow N5 \rightarrow N6 \rightarrow N11 \rightarrow N12$, the time taken to reach node $N12$ from node $N4$ will be $6.5\,ns$ and there were 3 intermediate nodes only. Thus the time needed with our selected path is reduced by,

$$\frac{1}{6.5} \times 100 = 15.38\%$$

Thus we have seen that there is no change in the intermediate nodes but the time taken is reduced by 15.38%.

The feasible schedule is only possible if all the nodes work properly. On the basis of our assumption the probability of the failure is given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

For scheduling the task from node $N4$ to node $N12$, the number of intermediate nodes crossed has been 3, which means $n = 3$. The expression evaluates to 0.27 which is lesser than the chances of failure observed in scheduling from node $N1$ to node $N12$.

The shortest path to all other nodes from node $N5$ is shown below

From	To	Distance	Route
5-N5	1-N1	1.5	5-2-1
5-N5	2-N2	0.5	5-2
5-N5	3-N3	1.5	5-2-3
5-N5	4-N4	1.0	5-4
5-N5	6-N6	1.5	5-6
5-N5	7-N7	3.0	5-8-7
5-N5	8-N8	1.0	5-8
5-N5	9-N9	1.5	5-8-9
5-N5	10-N10	3.5	5-2-3-10
5-N5	11-N11	3.5	5-6-11
5-N5	12-N12	4.5	5-8-9-12
5-N5	13-N13	5.0	5-2-1-18-17-13
5-N5	14-N14	4.0	5-2-1-18-17-14
5-N5	15-N15	4.0	5-2-1-18-17-15
5-N5	16-N16	4.0	5-2-1-18-17-16
5-N5	17-N17	3.0	5-2-1-18-17
5-N5	18-A	2.5	5-2-1-18
5-N5	19-B	4.5	5-2-3-10-19

Figure 4.6: Shortest paths from node $N5$ to all other nodes

From the results shown, it has been observed that it takes maximum time to reach node $N13$ from node $N5$ and that is $5\,ns$. The intermediate nodes which is needed to be crossed on this path are 4. If the path chosen to reach node $N13$ from node $N5$ other than our optimized path, $N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, then the time taken to reach node $N13$ from node $N5$ is $5.5\,ns$. But the number of intermediate nodes has been reduced in this path which will be just 3. In this case the optimized path selected has lesser delay but has to traverse 1 more node. The time taken in our optimized path as compared to the other arbitrary path taken is reduced by,

$$\frac{0.5}{5.5} \times 100 = 9.09\%$$

Thus our optimized path result in 9.09% reduction in the time taken with 1 extra node traversed. The failure of any one intermediate node results in the infeasible schedule which is not desirable. The probabilty of failure for n intermediate nodes is given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

From node $N5$ to node $N13$, the value of n for the optimized path is 4 and the expression evaluates to 0.34.

The following result displays shortest path from node $N6$ to all other nodes

From	To	Distance	Route
6-N6	1-N1	3.0	6-3-2-1
6-N6	2-N2	2.0	6-3-2
6-N6	3-N3	1.0	6-3
6-N6	4-N4	2.5	6-5-4
6-N6	5-N5	1.5	6-5
6-N6	7-N7	3.5	6-9-8-7
6-N6	8-N8	1.5	6-9-8
6-N6	9-N9	1.0	6-9
6-N6	10-N10	3.0	6-3-10
6-N6	11-N11	2.0	6-11
6-N6	12-N12	4.0	6-9-12
6-N6	13-N13	6.5	6-3-2-1-18-17-13
6-N6	14-N14	5.5	6-3-2-1-18-17-14
6-N6	15-N15	5.5	6-3-2-1-18-17-15
6-N6	16-N16	5.5	6-3-2-1-18-17-16
6-N6	17-N17	4.5	6-3-2-1-18-17
6-N6	18-A	4.0	6-3-2-1-18
6-N6	19-B	4.0	6-3-10-19

Figure 4.7: Shortest paths from node $N6$ to all other nodes

From the results, it has been observed that it takes maximum amount of time to reach node $N13$ from node $N6$ and also the maximum number of nodes have to be traversed. It takes $6.5\,ns$ to reach node $N13$ from node $N6$ and 5 intermediate nodes has to be crossed. If we take any other path, say, $N6 \rightarrow N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, the number of nodes in this path is reduced by 1 but the time taken is $7\,ns$. The time taken to reach node $N13$ from node $N6$, as compared to other arbitrary path selected, has been reduced by,

$$\frac{0.5}{7} \times 100 = 7.14\%$$

Hence the optimized path selected resulted in reduction of time taken by 7.14% with the penalty of 1 extra node traversed.

In order to ensure feasible schedule, the proper working of all the intermediate nodes is necessary. As we have ssen in other paths, with incresing number of nodes probabilty of failure increases and hence we aim at reducing the number of nodes. For the optimized path from node $N6$ to $N13$, the value of n is 5 and the expression $\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$, evaluates to 0.4. This increase in the value of probability is because of the increase in the number of intermediate nodes in comparison with the other cases discussed above.

The following results is for the shortest paths from all other nodes from node $N7$

From	To	Distance	Route
7-N7	1-N1	2.0	7-18-1
7-N7	2-N2	3.0	7-18-1-2
7-N7	3-N3	4.0	7-18-1-2-3
7-N7	4-N4	3.0	7-4
7-N7	5-N5	3.0	7-8-5
7-N7	6-N6	3.5	7-8-9-6
7-N7	8-N8	2.0	7-8
7-N7	9-N9	2.5	7-8-9
7-N7	10-N10	6.0	7-18-1-2-3-10
7-N7	11-N11	5.5	7-8-9-6-11
7-N7	12-N12	5.5	7-8-9-12
7-N7	13-N13	3.5	7-18-17-13
7-N7	14-N14	2.5	7-18-17-14
7-N7	15-N15	2.5	7-18-17-15
7-N7	16-N16	2.5	7-18-17-16
7-N7	17-N17	1.5	7-18-17
7-N7	18-A	1.0	7-18
7-N7	19-B	6.5	7-8-9-12-19

Figure 4.8: Shortest paths from node $N7$ to all other nodes

From the results shown above, it has been observed that to reach node B from $N7$, maximum amount of time has been taken and just 3 nodes were traversed in order to reach node B from node $N7$. It takes $6.5\,ns$ to reach node B from node $N7$. If some other path is taken, $N7 \rightarrow N8 \rightarrow N9 \rightarrow N6 \rightarrow N11 \rightarrow B$, the time taken would have been $7.5\,ns$ and the number of intermediate nodes will be 4. So, our optimized path has reduced the time taken to reach node B from node $N7$ as compared to other path selected by,

$$\frac{1}{7.5} \times 100 = 13.33\%$$

Thus our optimized path has resulted in the reduction of time taken by 13.33% along with the reduction of 1 intermediate node.

The optimized path to reach node B from node $N7$ has 3 intermediate nodes and all should work properly for the feasible schedule. The expression for the probability of failure given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

For the path from node $N7$ to node B, $n = 3$ and the expression given above evaluates to 0.27 which is same as that for the path to node $N12$ from nodes $N4$ and $N2$.

The following results shows the shortest path to all other nodes from node $N8$

From	To	Distance	Route
8-N8	1-N1	2.5	8-5-2-1
8-N8	2-N2	1.5	8-5-2
8-N8	3-N3	2.5	8-5-2-3
8-N8	4-N4	2.0	8-5-4
8-N8	5-N5	1.0	8-5
8-N8	6-N6	1.5	8-9-6
8-N8	7-N7	2.0	8-7
8-N8	9-N9	0.5	8-9
8-N8	10-N10	4.5	8-5-2-3-10
8-N8	11-N11	3.5	8-9-6-11
8-N8	12-N12	3.5	8-9-12
8-N8	13-N13	5.5	8-7-18-17-13
8-N8	14-N14	4.5	8-7-18-17-14
8-N8	15-N15	4.5	8-7-18-17-15
8-N8	16-N16	4.5	8-7-18-17-16
8-N8	17-N17	3.5	8-7-18-17
8-N8	18-A	3.0	8-7-18
8-N8	19-B	4.5	8-9-12-19

Figure 4.9: Shortest paths from node $N8$ to all other nodes

From the results, it has been observed that it takes maximum time to reach node $N13$ from node $N8$ and it has to cross 3 intermediate nodes. The time taken to reach node $N13$ from node $N8$ has been observed to be $5.5\,ns$. If some other path was taken to reach node $N13$ from node $N8$, $N8 \rightarrow N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, the time taken will be $6.5\,ns$ and the number of intermediate nodes is 4. The optimized path selected has the advantage of this arbitrary path in terms of time taken as well as the number of intermediate nodes traversed. The time taken to reach node $N13$ from node $N8$ in our optimized path has been reduced by,

$$\frac{1}{6.5} \times 100 = 15.38\%$$

Thus the reduction of 15.38% has been observed by our optimized path and it takes 1 less intermediate node to reach node $N13$ from node $N8$.

The probability of failure of the schedule with the assumption that probability of failure of each node is $\frac{1}{10}$, is given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

For the path from $N8$ to $N13$, n is equal to 3 and the expression evaluates to 0.27.

The following results displays the shortest paths to all other nodes from node $N9$

From	To	Distance	Route
9-N9	1-N1	3.0	9-8-5-2-1
9-N9	2-N2	2.0	9-8-5-2
9-N9	3-N3	2.0	9-6-3
9-N9	4-N4	2.5	9-8-5-4
9-N9	5-N5	1.5	9-8-5
9-N9	6-N6	1.0	9-6
9-N9	7-N7	2.5	9-8-7
9-N9	8-N8	0.5	9-8
9-N9	10-N10	4.0	9-6-3-10
9-N9	11-N11	3.0	9-6-11
9-N9	12-N12	3.0	9-12
9-N9	13-N13	6.0	9-8-7-18-17-13
9-N9	14-N14	5.0	9-8-7-18-17-14
9-N9	15-N15	5.0	9-8-7-18-17-15
9-N9	16-N16	5.0	9-8-7-18-17-16
9-N9	17-N17	4.0	9-8-7-18-17
9-N9	18-A	3.5	9-8-7-18
9-N9	19-B	4.0	9-12-19

Figure 4.10: Shortest paths from node $N9$ to all other nodes

From the results shown above, it has been observed that it takes maximum amount of time to reach node $N13$ from node $N9$. The time taken from node $N9$ to node $N13$ has been obseved to be $6\ ns$ and the number of intermediate nodes in the path is 4. If some other path is taken except the optimized path, $N9 \rightarrow N8 \rightarrow N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, the time taken is $7\ ns$ and the number of intermediate nodes are 5. The reduction in the time taken by our optimized path is given by,

$\frac{1}{7} \times 100 = 14.28\%$

Thus the reduction of 14.28% is observed along with the elimination of 1 extra intermediate node on going from node $N9$ to node $N13$.

The optimized path from node $N9$ to node $N13$ has 4 intermediate nodes ana all should function properly in order to have the efficient schedule. The expression for the probability of failure as discussed before is given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

For the selected optimized path, $n = 4$ and the probability of failure comes out to be 0.34.

The following results contain the shortest path from node $N10$ to all other nodes

From	To	Distance	Route
10-N10	1-N1	4.0	10-3-2-1
10-N10	2-N2	3.0	10-3-2
10-N10	3-N3	2.0	10-3
10-N10	4-N4	4.5	10-3-2-5-4
10-N10	5-N5	3.5	10-3-2-5
10-N10	6-N6	3.0	10-3-6
10-N10	7-N7	6.0	10-3-2-1-18-7
10-N10	8-N8	4.5	10-3-2-5-8
10-N10	9-N9	4.0	10-3-6-9
10-N10	11-N11	1.0	10-11
10-N10	12-N12	2.0	10-19-12
10-N10	13-N13	7.5	10-3-2-1-18-17-13
10-N10	14-N14	6.5	10-3-2-1-18-17-14
10-N10	15-N15	6.5	10-3-2-1-18-17-15
10-N10	16-N16	6.5	10-3-2-1-18-17-16
10-N10	17-N17	5.5	10-3-2-1-18-17
10-N10	18-A	5.0	10-3-2-1-18
10-N10	19-B	1.0	10-19

Figure 4.11: Shortest paths from node $N10$ to all other nodes

From the results, it has been observed that node $N13$ is farthest from node $N10$ and hence it takes maximum time to reach node $N13$ from node $N10$. It takes $7.5\,ns$ and 5 intermediate nodes to reach node $N13$ from node $N10$. If some other path is chosen, $N10 \rightarrow N3 \rightarrow N2 \rightarrow N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, the time taken will have been $9\,ns$ and number of intermediate nodes will be increased to 6. The optimized path has resulted in the reduction of time taken from the node $N10$ to node $N13$ and also the number of intermediate nodes has been reduced. The time taken to reach node $N13$ from node $N10$, in comparison with the arbitrary path chosen, has been reduced by,

$$\frac{1.5}{9} \times 100 = 16.67\%$$

Thus our optimized path resulted in the reduction of 16.67% in terms of time needed to reach node $N13$ from node $N10$.

The probability of failure of the schedule which is in accordance with our assumption is given by

$$\sum_{r=1}^{n} {}_{r}^{n}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

For the optimized path from node $N10$ to $N13$, $n = 5$ and the expression evaluates to 0.4.

The folowing results shows the shortest paths from node $N11$ to all other nodes

From	To	Distance	Route
11-N11	1-N1	5.0	11-6-3-2-1
11-N11	2-N2	4.0	11-6-3-2
11-N11	3-N3	3.0	11-6-3
11-N11	4-N4	4.5	11-6-5-4
11-N11	5-N5	3.5	11-6-5
11-N11	6-N6	2.0	11-6
11-N11	7-N7	5.5	11-6-9-8-7
11-N11	8-N8	3.5	11-6-9-8
11-N11	9-N9	3.0	11-6-9
11-N11	10-N10	1.0	11-10
11-N11	12-N12	2.0	11-12
11-N11	13-N13	8.5	11-6-3-2-1-18-17-13
11-N11	14-N14	7.5	11-6-3-2-1-18-17-14
11-N11	15-N15	7.5	11-6-3-2-1-18-17-15
11-N11	16-N16	7.5	11-6-3-2-1-18-17-16
11-N11	17-N17	6.5	11-6-3-2-1-18-17
11-N11	18-A	6.0	11-6-3-2-1-18
11-N11	19-B	2.0	11-19

Figure 4.12: Shortest paths from node $N11$ to all other nodes

From the results shown above, it has been observed node $N13$ can be reached in $8.5\,ns$ from node $N11$ and it is farthest from node $N11$. The number of intermediate nodes has been observed to be 6. If other path is chosen to node $N13$ from node $N11$, $N11 \rightarrow N6 \rightarrow N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, the number of intermediate nodes will remain 6 only but the time taken to reach node $N13$ from $N11$ will be $9\,ns$. The reduction in the time in our optimized path as compared to this path is given by,

$$\frac{0.5}{9} \times 100 = 5.56\%$$

Thus the reduction of 5.56% has been observed with our optimized path without an reduction in the number of intermediate nodes.

The expression for the probability of failure given by the expression,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

The value of this expression for the path from node $N11$ to $N13$ which has 6 intermediate nodes comes out to be 0.42, that is maximum value observed as compared to all the previous cases. So, as the number of nodes increased the chances of failing the optimal schedule has increased.

The following results shows the shortest paths to all other nodes from node $N12$

From	To	Distance	Route
12-N12	1-N1	6.0	12-9-8-5-2-1
12-N12	2-N2	5.0	12-9-8-5-2
12-N12	3-N3	4.0	12-19-10-3
12-N12	4-N4	5.5	12-9-8-5-4
12-N12	5-N5	4.5	12-9-8-5
12-N12	6-N6	4.0	12-9-6
12-N12	7-N7	5.5	12-9-8-7
12-N12	8-N8	3.5	12-9-8
12-N12	9-N9	3.0	12-9
12-N12	10-N10	2.0	12-19-10
12-N12	11-N11	2.0	12-11
12-N12	13-N13	9.0	12-9-8-7-18-17-13
12-N12	14-N14	8.0	12-9-8-7-18-17-14
12-N12	15-N15	8.0	12-9-8-7-18-17-15
12-N12	16-N16	8.0	12-9-8-7-18-17-16
12-N12	17-N17	7.0	12-9-8-7-18-17
12-N12	18-A	6.5	12-9-8-7-18
12-N12	19-B	1.0	12-19

Figure 4.13: Shortest paths from node $N12$ to all other nodes

From the results shown above, the following observations can be drawn:

Node $N13$ is farthest from node $N12$ and takes $9\,ns$ to reach from node $N12$ and the number of intermediate nodes encountered in the path is 5.

If any other path is taken from node $N12$, $N12 \rightarrow N9 \rightarrow N8 \rightarrow N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, the time taken to reach node $N13$ from node $N12$ will be $10\,ns$ and number of intermediate nodes will remain 6 only.

Reduction in the time taken with our optimized path as compared to the other selected path is given by,

$$\frac{1}{10} \times 100 = 10\%$$

Thus a reduction of 10% has been observed with the optimized path in comparison with the selected path.

The probability of failure of the schedule which is given by the expression,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

evaluates to be 0.4 for $n = 5$ through the path from $N12$ to $N13$.

The following results shows the shortest paths to all other nodes from node $N13$

From	To	Distance	Route
13-N13	1-N1	3.5	13-17-18-1
13-N13	2-N2	4.5	13-17-18-1-2
13-N13	3-N3	5.5	13-17-18-1-2-3
13-N13	4-N4	4.5	13-17-18-4
13-N13	5-N5	5.0	13-17-18-1-2-5
13-N13	6-N6	6.5	13-17-18-1-2-3-6
13-N13	7-N7	3.5	13-17-18-7
13-N13	8-N8	5.5	13-17-18-7-8
13-N13	9-N9	6.0	13-17-18-7-8-9
13-N13	10-N10	7.5	13-17-18-1-2-3-10
13-N13	11-N11	8.5	13-17-18-1-2-3-6-11
13-N13	12-N12	9.0	13-17-18-7-8-9-12
13-N13	14-N14	3.0	13-17-14
13-N13	15-N15	3.0	13-17-15
13-N13	16-N16	3.0	13-17-16
13-N13	17-N17	2.0	13-17
13-N13	18-A	2.5	13-17-18
13-N13	19-B	8.5	13-17-18-1-2-3-10-19

Figure 4.14: Shortest paths from node $N13$ to all other nodes

From the results shown above, the following observations has been drawn:

Node $N12$ is farthest from node $N13$ and it takes $9\,ns$ to reach node $N12$ from node $N13$ and the number of intermediate nodes traversed has been found to be 5.

If other path is chosen from node $N13$ to node $N12$, $N13 \rightarrow N17 \rightarrow A \rightarrow N4 \rightarrow N5 \rightarrow N6 \rightarrow N11 \rightarrow N12$, the time taken along this path is $11\,ns$ and number of intermediate nodes has been increased to 6. Reduction in the time needed to reach node $N12$ from node $N13$ is given by,

$$\frac{2}{11} \times 100 = 18.18\%$$

Thus a reduction of 18.18% has been observed with the optimized path in comparison with the selected path.

The error probability for the schedule from node $N13$ to node $N12$ given by the expression,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

evaluates to be equal to 0.4 for $n = 5$ and this value is same as that of the probability of failure along the path to node $N13$ from the nodes $N6$, $N10$ and $N12$. Thus node $N13$ is critical as compared to other nodes as the probability of failure is higher along 3 paths.

The following results depicts the shortest paths from node $N14$ to all other nodes

From	To	Distance	Route
14-N14	1-N1	2.5	14-17-18-1
14-N14	2-N2	3.5	14-17-18-1-2
14-N14	3-N3	4.5	14-17-18-1-2-3
14-N14	4-N4	3.5	14-17-18-4
14-N14	5-N5	4.0	14-17-18-1-2-5
14-N14	6-N6	5.5	14-17-18-1-2-3-6
14-N14	7-N7	2.5	14-17-18-7
14-N14	8-N8	4.5	14-17-18-7-8
14-N14	9-N9	5.0	14-17-18-7-8-9
14-N14	10-N10	6.5	14-17-18-1-2-3-10
14-N14	11-N11	7.5	14-17-18-1-2-3-6-11
14-N14	12-N12	8.0	14-17-18-7-8-9-12
14-N14	13-N13	3.0	14-17-13
14-N14	15-N15	2.0	14-17-15
14-N14	16-N16	2.0	14-17-16
14-N14	17-N17	1.0	14-17
14-N14	18-A	1.5	14-17-18
14-N14	19-B	7.5	14-17-18-1-2-3-10-19

Figure 4.15: Shortest paths from node $N14$ to all other nodes

From the results shown above, the following observations has been drawn:

Node $N12$ is farthest and it takes $8\,ns$ to reach there from node $N14$.

The number of intermediate nodes that has to be crossed to reach node $N12$ from node $N14$ is 5.

If another path is chosen to reach node $N12$ from node $N14$, $N14 \rightarrow N17 \rightarrow A \rightarrow N4 \rightarrow N5 \rightarrow N6 \rightarrow N11 \rightarrow N12$, the time taken in this path has been found to be $10\,ns$ and the number of intermediate nodes has been increased to 6. Reduction in the time required to reach node $N12$ from node $N14$ has been found to be,

$$\frac{2}{10} \times 100 = 20\%$$

Thus a reduction of 20% in the time required has been achieved by our optimized path in comparison to the selected path.

The expression for the probability of failure having n intermediate nodes given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

evaluates to 0.4 for the selected optimized path as the value of n along the path is 5. Thus, the proabability of failure is same as that along the path from $N13$ to node $N12$ as observed before.

The following results gives the shortest path from node $N15$ to all other nodes

From	To	Distance	Route
15-N15	1-N1	2.5	15-17-18-1
15-N15	2-N2	3.5	15-17-18-1-2
15-N15	3-N3	4.5	15-17-18-1-2-3
15-N15	4-N4	3.5	15-17-18-4
15-N15	5-N5	4.0	15-17-18-1-2-5
15-N15	6-N6	5.5	15-17-18-1-2-3-6
15-N15	7-N7	2.5	15-17-18-7
15-N15	8-N8	4.5	15-17-18-7-8
15-N15	9-N9	5.0	15-17-18-7-8-9
15-N15	10-N10	6.5	15-17-18-1-2-3-10
15-N15	11-N11	7.5	15-17-18-1-2-3-6-11
15-N15	12-N12	8.0	15-17-18-7-8-9-12
15-N15	13-N13	3.0	15-17-13
15-N15	14-N14	2.0	15-17-14
15-N15	16-N16	2.0	15-17-16
15-N15	17-N17	1.0	15-17
15-N15	18-A	1.5	15-17-18
15-N15	19-B	7.5	15-17-18-1-2-3-10-19

Figure 4.16: Shortest paths from node $N15$ to all other nodes

From the results shown above, the observations that has been drawn are:

The time taken to reach node $N12$ is maximun from node $N15$ and it has been evaluated to be $8\,ns$. The number of intermediate nodes has been found to be 5 along this path.

If we take another path to reach node $N12$ from node $N15$, $N15 \rightarrow N17 \rightarrow A \rightarrow N4 \rightarrow N5 \rightarrow N6 \rightarrow N11 \rightarrow N12$, the time taken to reach node $N12$ from node $N15$ on this path has been observed to be $10\,ns$. The intermediate nodes that needs to be crossed to reach node $N12$ from node $N15$ has been observed to be 6. The reduction in the time taken by the optimized path as compared to the other selected path has been found to be,

$$\frac{2}{10} \times 100 = 20\%$$

Thus a reduction of 20% in the time required has been achieved by our optimized path in comparison to the selected path and also it resulted in the reduction of 1 intermediate node.

The expression for the probability of failure of schedule along the path given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

evaluates to 0.4 for $n = 5$ along the selected path which is same as observed in previous case.

The following results depict the shortest path to all other nodes from node $N16$

From	To	Distance	Route
16-N16	1-N1	2.5	16-17-18-1
16-N16	2-N2	3.5	16-17-18-1-2
16-N16	3-N3	4.5	16-17-18-1-2-3
16-N16	4-N4	3.5	16-17-18-4
16-N16	5-N5	4.0	16-17-18-1-2-5
16-N16	6-N6	5.5	16-17-18-1-2-3-6
16-N16	7-N7	2.5	16-17-18-7
16-N16	8-N8	4.5	16-17-18-7-8
16-N16	9-N9	5.0	16-17-18-7-8-9
16-N16	10-N10	6.5	16-17-18-1-2-3-10
16-N16	11-N11	7.5	16-17-18-1-2-3-6-11
16-N16	12-N12	8.0	16-17-18-7-8-9-12
16-N16	13-N13	3.0	16-17-13
16-N16	14-N14	2.0	16-17-14
16-N16	15-N15	2.0	16-17-15
16-N16	17-N17	1.0	16-17
16-N16	18-A	1.5	16-17-18
16-N16	19-B	7.5	16-17-18-1-2-3-10-19

Figure 4.17: Shortest paths from node $N16$ to all other nodes

From the results shown above, the various observations which has been drawn are given below:

In this case also, node $N12$ is farthest from node $N16$ and it has been observed that the time taken to reach node $N12$ from node $N16$ is $8\,ns$ with 5 intermediate nodes.

If some other path is chosen from node $N16$ to node $N12$, $N16 \rightarrow N17 \rightarrow A \rightarrow N4 \rightarrow N5 \rightarrow N6 \rightarrow N11 \rightarrow N12$, the time needed to reach node $N12$ from node $N16$ is increased to $10\,ns$. The number of intermediate nodes has been increased to 6. The reduction in the time taken by the optimized path as compared to the other selected path has been found to be,

$$\frac{2}{10} \times 100 = 20\%$$

Thus a reduction of 20% in the time required has been achieved by our optimized path in comparison to the selected path and also it resulted in the reduction of 1 intermediate node.

The expression for the probability of failure of the schedule given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

evaluates to 0.4 ($n = 4$) along the selected path which is again same as observed in the last case. Thus schedule to node $N12$ from nodes $N13$, $N14$, $N15$ and $N16$ are equally probable to fail.

The following result shows the shortest path to all other nodes from node $N17$

From	To	Distance	Route
17-N17	1-N1	1.5	17-18-1
17-N17	2-N2	2.5	17-18-1-2
17-N17	3-N3	3.5	17-18-1-2-3
17-N17	4-N4	2.5	17-18-4
17-N17	5-N5	3.0	17-18-1-2-5
17-N17	6-N6	4.5	17-18-1-2-3-6
17-N17	7-N7	1.5	17-18-7
17-N17	8-N8	3.5	17-18-7-8
17-N17	9-N9	4.0	17-18-7-8-9
17-N17	10-N10	5.5	17-18-1-2-3-10
17-N17	11-N11	6.5	17-18-1-2-3-6-11
17-N17	12-N12	7.0	17-18-7-8-9-12
17-N17	13-N13	2.0	17-13
17-N17	14-N14	1.0	17-14
17-N17	15-N15	1.0	17-15
17-N17	16-N16	1.0	17-16
17-N17	18-A	0.5	17-18
17-N17	19-B	6.5	17-18-1-2-3-10-19

Figure 4.18: Shortest paths from node $N17$ to all other nodes

The following observations has been drawn from the results shown above:

In this case also, the node $N12$ has been found to be farthest from node $N17$ and it takes $7\,ns$ to reach node $N12$ from node $N17$ with 4 intermediate nodes along the optimized path.

If other path is taken from node $N17$ to node $N12$, $N17 \rightarrow A \rightarrow N4 \rightarrow N5 \rightarrow N6 \rightarrow N11 \rightarrow N12$, the time taken to reach node $N12$ from node $N17$ has been observed to be $9\,ns$. The reduction in the time taken to reach node $N12$ from node $N17$ has been observed to be,

$$\frac{2}{9} \times 100 = 22.22\%$$

Thus a reduction of 22.22% in terms of time taken has been observed by our optimized path and the number of intermediate node which has to be traversed is reduced by 1.

Now, the expression for the probability of failure for the schedule given by,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

evaluates to 0.34 for $n = 4$ along the selected path which is same as the probability of failure of the schedule from node $N1$ to node $N12$. Hence the paths to node $N12$ from nodes $N1$ and $N16$ are equally probable to fail as the number of intermediate nodes are same in both cases.

The following results give the shortest path from node A to all other nodes

From	To	Distance	Route
18-A	1-N1	1.0	18-1
18-A	2-N2	2.0	18-1-2
18-A	3-N3	3.0	18-1-2-3
18-A	4-N4	2.0	18-4
18-A	5-N5	2.5	18-1-2-5
18-A	6-N6	4.0	18-1-2-3-6
18-A	7-N7	1.0	18-7
18-A	8-N8	3.0	18-7-8
18-A	9-N9	3.5	18-7-8-9
18-A	10-N10	5.0	18-1-2-3-10
18-A	11-N11	6.0	18-1-2-3-6-11
18-A	12-N12	6.5	18-7-8-9-12
18-A	13-N13	2.5	18-17-13
18-A	14-N14	1.5	18-17-14
18-A	15-N15	1.5	18-17-15
18-A	16-N16	1.5	18-17-16
18-A	17-N17	0.5	18-17
18-A	19-B	6.0	18-1-2-3-10-19

Figure 4.19: Shortest paths from node A to all other nodes

From the results shown above, the following observations has been drawn:

In this case again, node $N12$ is farthest from node A and it takes $6.5\,ns$ to reach node $N12$ from node A with 3 intermediate nodes along the path.

If any other path has been taken to reach node $N12$ from node A, $A \rightarrow N4 \rightarrow N5 \rightarrow N6 \rightarrow N11 \rightarrow N12$, the time taken to reach node $N12$ from node A has been increased to $8.5\,ns$ and intermediate nodes has been increased to 4.

The reduction in time required to reach node $N12$ from node A has been evaluated to be,

$$\frac{2}{8.5} \times 100 = 23.53\%$$

Thus optimized path resulted in the reduction of 23.53% in terms of time taken from node A to node $N12$ and the number of intermediate nodes has been reduced by 1.

The probability of error for the schedule between 2 nodes which is given by the expression,

$$\sum_{r=1}^{n} {}^{n}_{r}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

evalutes to be 0.27 as the value of n is 3 along the selected path. Hence the scheduling the task to node $N12$ from nodes $N2$, $N4$ and A are equally probable to fail on the failure of even 1 node.

The following results give the shortest path from node B to all other nodes

From	To	Distance	Route
19-B	1-N1	5.0	19-10-3-2-1
19-B	2-N2	4.0	19-10-3-2
19-B	3-N3	3.0	19-10-3
19-B	4-N4	5.5	19-10-3-2-5-4
19-B	5-N5	4.5	19-10-3-2-5
19-B	6-N6	4.0	19-10-3-6
19-B	7-N7	6.5	19-12-9-8-7
19-B	8-N8	4.5	19-12-9-8
19-B	9-N9	4.0	19-12-9
19-B	10-N10	1.0	19-10
19-B	11-N11	2.0	19-11
19-B	12-N12	1.0	19-12
19-B	13-N13	8.5	19-10-3-2-1-18-17-13
19-B	14-N14	7.5	19-10-3-2-1-18-17-14
19-B	15-N15	7.5	19-10-3-2-1-18-17-15
19-B	16-N16	7.5	19-10-3-2-1-18-17-16
19-B	17-N17	6.5	19-10-3-2-1-18-17
19-B	18-A	6.0	19-10-3-2-1-18

Figure 4.20: Shortest paths from node B to all other nodes

From the results shown above, following observations have been made:

Node $N13$ is farthest from node B and it takes $8.5\ ns$ to reach from node B to node $N13$.

The number of intermediate nodes traversed along the optimized path has been found to be 6.

If any other path is taken from node B to reach node $N13$, $B \rightarrow N11 \rightarrow N6 \rightarrow N5 \rightarrow N4 \rightarrow A \rightarrow N17 \rightarrow N13$, the time taken has been observed to be increased to $11\ ns$ and number of intermediate nodes remained 6 only. The reduction in the time observed in the optimzed path is given by,

$$\frac{2.5}{11} \times 100 = 22.72\%$$

Thus optimized path resulted in the reduction of 22.72% in terms of time taken from node B to node $N13$.

The expression for the probability of failure for the schedule with n intermediate nodes given by,

$$\sum_{r=1}^{n} {}_{r}^{n}C \left(\frac{1}{10}\right)^{r} \left(\frac{9}{10}\right)^{n-r}$$

evaluates to 0.42 for the selected path to node $N13$ from node B as $n = 6$. The more the number of intermediate nodes, the more is the chances of failure of schedule. Scheduling the task to node $N13$ from nodes $N11$ and B are equally probable to fail as observed above.

The following results have been drawn from the the previous section

The shortest paths between all the nodes has been evaluated along with the calculation of probability of error in all the paths. It has been observed that node $N13$ and node $N12$ are critical nodes as maximum amount of time is required to reach these nodes from most of the nodes. From nodes $N1$, $N2$, $N4$, $N13$, $N14$, $N15$, $N16$, $N17$ and A, a task takes maximum amount of time to reach node $N12$. The data from node $N7$ takes maximum time to reach node B and all the remaining nodes take maximum time to reach node $N13$. The above observations justify that nodes $N12$ and $N13$ are the most critical nodes and optimization of paths is highly required for these two nodes. This optimization has been achieved by the technique devised and the delay has been minimized by this optimized path as compared to other paths.

The probability of failure of schedule because of the failure of intermediate nodes has been evaluated and it has been found that scheduling the tasks to nodes $N13$ and $N12$ is more likely to fail. It has also been observed that as the number of intermediate nodes increases, the probability of failure increases which in turn makes the schedule along the path less desirable. The optimization of the paths and the evaluation of the probability of failure have suggested that least critical tasks should be scheduled to the nodes $N12$ and $N13$ so that the critical tasks don't miss their deadlines and the feasible schedule is obtained.

Chapter 5 Conclusions and Future Scope

The study of various real time systems and their classifications has been done. The scheduling algorithms have also been discussed in detail along with the comparison of different real time scheduling algorithms. The implementation of shortest path algorithm in real time has been done for the tasks. The problem for finding the shortest path in the communication network has been taken and simulated accordingly. Evaluation of shortest path and the probability of error for different nodes have been done. Resource optimization has been achieved in our technique as it only involves the optimal paths between two nodes.

The work can be extended by exploiting the priority constraints for static as well as dynamic priorities. The response of the system becomes non-deterministic with dynamic priorities. The additional overloads that will be incurred on the system due to dynamic priorities will have to be considered in the calculation of delay. The limited pre-emption techniques can be exploited for the efficient working of real-time systems. The overheads can also be managed by using partly pre-emptive systems. The techniques can be tested on the different platforms or software so as to ensure its optimality for all systems.

List of Publications

- Ishan Khera, Ajay Kakkar, "Comparative Study of Scheduling Algorithms for Real Time Environment", International Journal of Computer Applications, Vol. 44, No. 2, 2012, pp. 5-8.
- Ishan Khera, Ajay Kakkar, "Scheduling Techniques for Resource Optimization in Real Time Environment", International Conference on Advancements in Computing and Communication, Vol. 2, 2012, pp. 482-486.

References

[1] Sanjoy K. Baruah and Joel Goossens, "Rate-Monotonic Scheduling on Uniform Multiprocessors", IEEE Transactions on Computers, Vol. 52, No. 7, 2003, pp. 966-970.

[2] Peng Li and Binoy Ravindran, "Fast, Best-Effort Real-Time Scheduling Algorithms", IEEE Transactions on Computers, Vol. 53, No. 9, 2004, pp. 1159-1175.

[3] Xiaoying Wang, Hai Zhao, Wenbo Zhang and Zhenyu Yin, "Preemptive Behavior Analysis and Improvement of Priority Scheduling Algorithms", 19th IEEE International Parallel and Distributed Processing Symposium, 2005, pp. 130-137.

[4] Chen Tianzhou, Hu Wei, Xie Bin and Yan Like, "A Real-Time Scheduling Algorithm for Embedded Systems with Various Resource Requirements", International Workshop on Networking, Architecture and Storages, 2006, pp. 43-46.

[5] Jiajing Zhuo, Chen Meng and Minghu Zou, " A Task Scheduling Algorithm of Single Processor Parallel Test System", 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Vol. 1, 2007, pp. 627- 632.

[6] S. Roman, H. Mecha, D. Mozos and J. Septien, "Constant complexity scheduling for hardware multitasking in two dimensional reconfigurable field-programmable gate arrays", IET Computers and Digital Techechniques, Vol. 2, No. 6, 2008, pp. 401–412.

[7] Pravanjan Choudhury, Rajeev Kumar and P.P. Chakrabarti, "Hybrid Scheduling of Dynamic Task Graphs with Selective Duplication for Multiprocessors under Memory and Time Constraints" IEEE Transactions on Parallel and Distributed Systems, Vol. 19, No. 7, 2008, pp. 967-980.

[8] Euiseong Seo, Jinkyu Jeong, Seonyeong Park and Joonwon Lee., "Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors", IEEE Transactions on Parallel and Distributed Systems, Vol. 19, No. 11, 2008, pp 1540-1552.

[9] Ming Xiong, Song Han, Kam-Yiu Lam and Deji Chen, "Deferrable Scheduling for Maintaining Real-Time Data Freshness: Algorithms, Analysis, and Results", IEEE Transactions on Computers, Vol. 57, No. 7, 2008, pp. 952-964.

[10] Fengxiang Zhang and Alan Burns, "Schedulability Analysis for Real-Time Systems with EDF Scheduling", IEEE Transactions on Computers, Vol. 58, No. 9, 2009, pp. 1250-1258.

[11] Enrico Bini, Thi Huyen, Chau Nguyen, Pascal Richard and Sanjoy K. Baruah, "A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines", IEEE Transactions on Computers, Vol. 58, No. 2, 2009, pp. 279-286.

[12] Marko Bertogna, Michele Cirinei, and Giuseppe Lipari, "Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms", IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 4, 2009, pp. 553-566.

[13] Wann-Yun Shieh and Bo-Wei Chen "Energy-Efficient Tasks scheduling Algorithm for Dual-core Real-time Systems", International Computer Symposium, 2010, pp. 568-575.

[14] Marko Bertogna and Sanjoy Baruah, "Limited Preemption EDF Scheduling of Sporadic Task Systems", IEEE Transactions on Industrial Informatics, Vol. 6, No. 4, 2010, pp. 579-591.

[15] Xian-Bo He, Gang-Yuan Zhang, Min Liu, Yu-Ping Zhao and Wei Li "A Fuzzy EDF Scheduling Algorithm Being Suitable for embedded Soft Real-time Systems in the Uncertain Environments", 2nd International Conference on Advanced Computer Control (ICACC), Vol. 5, 2010, pp. 583-587.

[16] A. Burns, R.I. Davis, P. Wang and F. Zhang "Partitioned EDF scheduling for multiprocessors using a C = D task splitting scheme", Journal of Real Time Systems, Vol. 48, No. 1, 2011, pp. 3-33.

[17] Wan Yeon Lee, "Energy-Efficient Scheduling of Periodic Real-Time Tasks on Lightly Loaded Multicore Processors", IEEE Transactions on Parallel and Distributed Systems, Vol. 23, No. 3, 2012, pp. 530-537.

[18] Alan Burns and Sanjoy Baruah, "Sustainability in Real-time Scheduling", Journal of Computing Science and Engineering, Vol. 2, No. 1, 2008, pp. 74-97.

[19] Giorgio C. Buttazzo, "Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications", Springer, Third edition.

[20] Eric Brewer, "Lottery Scheduling", Advanced Topics in Computer Systems, www.cs.berkeley.edu/~brewer/cs262/Lec-scheduling.pdf.

[21] Franco Callari, "Types of Scheduling - Long Term and Medium Term Scheduling", http://www.cim.mcgill.ca/~franco/OpSys-304-427/lecture-notes/node38.html.

[22] Howard Hamilton, Kimberly Lemieux, Allan Tease, "Scheduling of Processes", http://www2.cs.uregina.ca/~hamilton/courses/330/notes/scheduling/scheduling.html.

[23] "CPU Scheduling", http://www.os-concepts.thiyagaraaj.com/cpu-process-scheduling.

[24] Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel", O'Reilly Online Catalogue, 2000.

[25] Hermann Kopetz, "Real-Time Systems: Design Principles for Distributed Embedded Applications", Springer, second edition.

[26] Peter Brucker, "Scheduling Algorithms", Springer, fifth edition.

[27] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan., "Optimization and approximation in deterministic sequencing and scheduling: A survey", Annals of Discrete Mathematics, 1979, pp. 287–326.

[28] C.L. Liu and James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment", Journal of the Association of Computing Machinery, Vol. 20, No. 1, 1973, pp. 46-61.

[29] Albert M. K. Cheng, "Real-Time Systems: Scheduling, Analysis, and Verification", Wiley Interscience, 2002.

[30] John A. Stankovic, Marco Spuri, Krithi Ramamritham and Giorgio C. Buttazzo, "Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms", Kluwer Academic Publishers, 1998.

[31] Omar U. Pereira Zapata and Pedro Mejia Alvarez, " EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation", delta.cs.cinvestav.mx/~pmalvarez/multitechreport.pdf.

[32] "The Concise Handbook of Real-Time Systems", TimeSys Corporation, Version 1.3, 2002.