ISSN:0975-9646

Instantaneous Utilization Based Scheduling Algorithms for Real Time Systems

Radhakrishna Naik[1], R.R.Manthalkar[2]
Pune University[1], SGGS Nanded[2]

Abstract –
This paper proposes a new novel scheduling algorithms to schedule periodic tasks for soft real time system. This is a planning based offline scheduler where tasks are scheduled on the basis of its instantaneous utilization. Here after every quantum of execution, instantaneous utilization of each task is calculated. Task which is having highest instantaneous utilization is scheduled to the processor. Since Instantaneous utilization factor(IUF) is temporarily variant factor, the priority of each task will vary continuously. Also It is often more desirable to complete some portions of every task rather than giving up completely the processing of some tasks. The Imprecise Computation Model was introduced to allow for the trade-off of the quality of computations in favor of meeting the deadline constraints. It is observed that scheduling performance metrics such as schedulability, CPU utilization, context switching, response time and reliability are improved by this approach as compared to scheduling algorithms such as RM, EDF, LLF, MUF scheduling algorithms.

KEYWORDS:
RM: Rate monotonic, EDF: Earliest deadline first, LLF: Least laxity first, MUF: Maximum urgency first, IUF: Instantaneous utilization first, IRIS: Increased reward with increased service. MIUF: Modified instantaneous scheduling algorithm.

I. INTRODUCTION

As soft real-time systems are less restrictive in nature, it is required that the critical processes receive priority over less critical ones. In a soft real-time system, missing some deadlines will lower the performance of the system, but system will still continue to operate. Preemptive schedulers usually offer better overall system utilization. So they are preferred over non preemptive schedulers. Deadline parameter of a request is the amount of time given to the system to execute and complete the request after it is arrived. Preemptive real-time scheduling algorithms can be broadly classified into two categories: static priority and dynamic priority. This classification is based on the manner in which priorities are assigned to tasks. A scheduling algorithm is said to be static if priorities are assigned to tasks a priori and do not change during run-time, for example Rate Monotonic (RM) Algorithm. A scheduling policy is said to be dynamic if the priorities of a task might change from request to request, for example Earliest Deadline First (EDF) algorithm. Static priority and dynamic priority scheduling has received tremendous attraction after pioneering work done by Liu and Layland [1]. RM Assigns priority of each task according to its period, so that the shorter periods get the higher priority. Liu and Layland have also found stronger utilization for a dynamic priority assignment policy called EDF. A task is assigned highest priority if its deadline is nearest and will be assigned lowest priority if deadline is farthest. The problem with RM scheduling algorithms is that-

- Unfortunately, with static scheduling, resources must be allocated pessimistically and scheduled under the assumption that interrupts occur at the maximum rate. When they do not, utilization is effectively reduced because unused resources cannot be reallocated.
- As the priorities cannot be changed easily at run-time, allocations must be based on worst-case conditions. Thus, if an operation requires 8 msec, static scheduling analysis must assume that 8 msec will be required for every invocation. Again, utilization is effectively penalized because the resource will be idle for 3 msec in the usual case.
- From a scheduling perspective, the main advantage of EDF and least laxity first (LLF) is that they overcome the utilization limitations of RM. In particular, the utilization phasing penalty. This is because EDF and LLF prioritize operations according to their dynamic run-time characteristics. They handle harmonic and non-harmonic periods comparably, and respond flexibly to invocation-to-invocation variations in resource requirements, allowing CPU time one operation does not use to be reallocated to other operations. Thus, they can produce schedules that are optimal in terms of CPU utilization.
- Purely dynamic scheduling approaches like LLF and EDF potentially relieve the utilization limitations of the static RM approach. However, they have a higher cost to evaluate the scheduling algorithm at run-time. In addition, these purely dynamic scheduling strategies offer no control over which operations will miss their deadlines if the schedulable bound is exceeded. As operations are added to the schedule to achieve higher utilization, the margin of safety for all operations decreases. Therefore, the risk of missing a deadline increases for every operation as the system becomes overloaded.

We have designed the scheduler by considering simple utilization based schedulability analysis technique. Our paper proposes dynamically changing priority based pre-emptive scheduling algorithm based on the instantaneous utilization of the task. Instantaneous utilization factor (IUF) is the processor utilization of the task at any instant. Priority of the task is based on this IUF of each task in the given task set. Since the IUF is the temporarily variant factor, the priority of each task varies continuously. Algorithm like maximum utilization first [MUF]/ maximum urgency first [MUF] have also dwelled on the idea of

scheduling according to utilization of the task . But this algorithm has considered the static utilization of the task rather than their instantaneous utilization value. IUF has number of interesting characteristics. The IUF maximizes utilization bound of schedule and has the capability to accommodate larger task set. IUF has capability of dynamic predictability i.e. we can predict the status of our scheduler at any instant of time in the future and whether the schedule will still remain feasible.

This paper suggests two algorithms

1. Instantaneous utilization first real time scheduling algorithm.
2. Modified IUF real time scheduling algorithm.

Thus contribution of this paper is to enhance context switching, response time and CPU utilization than that of IUF scheduling algorithm.

II. RELATED WORK

Liu and Layland [1] were perhaps the first to formally study priority driven algorithms. They focused on the problem of scheduling periodic tasks on a single processor and proposed two preemptive algorithms, RM and EDF. RM algorithm assigns priorities to tasks based on their period. Higher the priority with shorter periods. Maximum utilization bound achieved is 69 %.This often referred as schedulability test. This test is true if period of invocation of task is equal to relative deadline of tasks. Inverse deadline allows a weakening of the condition which requires equality between periods and deadlines in static-priority schemes. The inverse deadline algorithm [2] assigns priorities to tasks according to their deadlines: the task with the shortest relative deadline is assigned the highest priority. Inverse deadline is optimal in the class of fixed-priority assignment algorithms in the sense that if any fixed-priority algorithm can schedule a set of tasks with deadlines shorter than periods, than inverse deadline will also schedule that task set. The computation given in the previous section can be extended to the case of two tasks with deadlines shorter than periods, scheduled with inverse deadline.

With dynamic priority assignment algorithms, priorities are assigned to tasks based on dynamic parameters that may change during task execution. The most important algorithms in this category are *earliest deadline first* [1] and *least laxity first* [3,4] The Maximum Urgency First (MUF) [5] scheduling algorithm supports both the deterministic rigor of the static RM scheduling approach and the flexibility of dynamic scheduling approaches such as EDF and MLF. RM assigns all priority components statically and EDF/LLF assigns all priority components dynamically. In contrast, MUF can assign both static and dynamic priority components.

Priorities can be assigned statically or dynamically based on different criteria like deadline, criticality, periodicity etc. Dynamic scheduling can be preemptive or non-preemptive K. Ramamritham, J.A Stankovic [6] work on 4-scheduling paradigms. Static table driven approaches are applicable to tasks that are periodic. Tables are constructed by using heuristics that identify the state and completion times of each task and tasks are dispatched according to this table. This is highly predictable approach, but is highly inflexible since any change to the tasks and their characteristics may require a complete overhaul of the table.

Utilization bound tests were first proposed Goossens e*t al.* [7] in which it is assumed that tasks have relative deadline equal to their period. Baker [8] slightly modified the assumption such that utilization bound test can be performed on tasks with relative deadline less than or equal to their period. Baker derived simple sufficient conditions for schedulability of systems of periodic or sporadic tasks in a multiprocessor preemptive scheduling environment.

Imprecise Computation Model was introduced [9] to allow for the trade-off of the quality of computations in favor of meeting the deadline constraints. In this model, a task is logically decomposed into two subtasks, mandatory and optional. The mandatory subtask of each task is required to be completed by its deadline, while the optional subtask can be left unfinished. If a task has an unfinished optional subtask, it incurs an error equal to the execution time of its unfinished portion. The Imprecise Computation Model is designed to model an iterative algorithm, where the task initially spends some time for initialization (the mandatory subtask) and then iterates to improve the quality of the solution (the optional subtask). Since the optional subtask corresponds to iterations to improve the quality of the solution, it can be left unfinished and still obtain a solution with a somewhat inferior quality. In this way, we can trade off the quality of the computation in favor of meeting deadlines. Therefore we considered quantum size equal to its mandatory portion and scheduled according to instantaneous utilization and optional portion is scheduled according to shortest job first criteria.

In this paper we propose that ,if in case there is fault, while executing mandatory portion, it has been suggested to provide redundancy to the mandatory portion as the faults of interest are those that are transient. Castillo e*t al.* [10] in their study of several systems indicates that the occurrences of transient faults are 10 to 50 times more frequent than permanent faults. In some applications this frequency can be quite large; one experiment on a satellite system observed 35 transient faults in a 15 minute interval due to cosmic ray ions [11]. To provide the flexibility needed to program fault tolerance, fixed priority preemptive scheduling suggested by A. Campbell et *al.* [12] can be used.

Arshad Iqbal and Asia Zafar[13] suggest that the design and analysis of a new scheduling algorithm. Dynamic Queue Deadline First (DQDF) to handle scheduling of dynamic multiple tasks in real time systems. They provide a approach that reduced the dead-line missing ratio but it has a higher CPU overhead.

III. INSTANTANEOUS UTILIZATION SCHEDULING ALGORITHM FRAMEWORK

A. Contribution of this paper

In this paper, the problem of predicting missing of deadline at any instant of time, low CPU overhead, justification of all tasks, no preemption and higher schedulability are addressed.
First we propose a framework of IUF scheduling algorithm, we run the case study for the same and finally we perform comparative evolution of RM , EDF,LLF and IUF.

B. System model

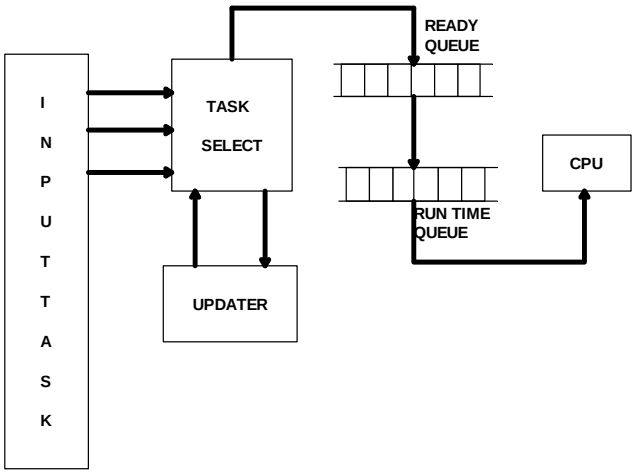

Figure 1.Architecture of IUF framework

This is basically a planning based scheduling algorithm where priorities are assigned based on Instantaneous Utilization .Every task is schedule for fix quantum of time .Planning of scheduling is considered for first iteration of period of invocation of the tasks. The quantum for which task is applied to CPU is Q_i. The total sum of quantum of all task (for that quantum iteration only) is $\sum Q_i=Q$.
In this framework, for given task set ,PTC is calculated by-
PTC =LCM{period of invocation of all tasks }
Now for time span equal to PTC tasks are mapped to CPU in following steps-
Step1:- Initially for given task set, calculate CPU utilization of each task using formula

$$U_0^i = C_0^i / P_0^i \qquad (1)$$

U_0^i =Initial utilization of i^{th} task.

C_0^i =Initial computation time.

P_0^i =Initial period of invocation.

Based on utilization [U_0^i] the task which is having higher value of Utilization is mapped for the CPU.
Step2:- Now in a given PTC, one task has executed for one quantum of time. Again calculate value of C_1^i , P_1^i by using following formula-

$$C_1^i = C_0^i - Q_i . \qquad (2)$$

$$P_1^i = P_0^i - Q . \qquad (3)$$

Where $\sum Q_i=Q$.

Then calculate new Instantaneous Utilization factor for i^{th} task for using formula 1 and 2

$$U_1^i = C_1^i / P_1^i . \qquad (4)$$

Where ,

C_1^i =Instantaneous computation time for i^{th} task

P_1^i = Instantaneous period of execution of i^{th} task.

U_1^i =Instantaneous Utilization of i^{th} task.

For second iteration of time Derive the table (T_i, C_1^i , P_1^i , U_1^i)

Again the task which is having highest instantaneous utilization will be having highest priority of execution for second iteration quantum.
Like wise, calculate

$$C_j^i = C_{j-1}^i - Q_i . \qquad (5)$$

$$P_j^i = P_{j-1}^i - Q . \qquad (6)$$

Calculate U_j^i using equation 4 and 5

$$U_j^i = C_j^i / P_j^i . \qquad (7)$$

Where,

U_j^i =Instantaneous utilization of i^{th} task for the j^{th} iteration of quantum

j = PTC end point.

Step 3: Hence task sequence in first PTC is derived. It is observed that at every step we can check whether the instantaneous utilization is less than initial utilization U_0^i .If at any given instant of time, it is observed that it is greater than U_0^i , it means that task is going to miss its deadline.
It is observed that in first PTC span , there is higher context switching between the task. In order to avoid context switching we are suggesting concept of run time data structure (run time

queue).Tasks are shuffled in run time queue as shown in figure 2.

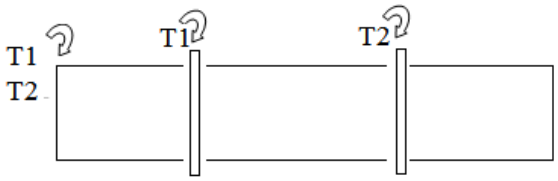

Figure 2. Task Shuffling

Initially in PTC span, consider time span between T_0 and T_1 shuffling the quantum of tasks which are similar. Then consider time span between T_1 and T_2 again shuffle the quantum of tasks which are similar. Lastly consider time span between T_2 and PTC –end and shuffle the quantum of tasks.

C. A Case Study: IUF Scheduling Algorithm

Consider following task set

T_1= (3, 9), T_2= (5, 11), T_3= (7, 38)

1. Calculate initial utilization using equation (1)

$$U_0^i = C_0^i / P_0^i$$

TABLE I. INITIAL TASK SET.

T_i	C_i	P_i	U_i
T_1	3	9	0.33
T_2	5	11	0.45
T_3	7	38	0.18

2. After first step, in Table 1 we observed that T_2 has the higher initial utilization so is mapped for execution.

For 1 Quantum

Calculate the new values of C_1, P_1, and U_1 using the following value:

C_0^1=3 P_0^1=9 Q_1=0

C_0^2=5 P_0^2=11 Q_2=1

C_0^3=7 P_0^3=38 Q_3=0

$Q=\sum (Q_i) =0+1+0=1$

Now using the formula,

$$C_1^i = C_0^i - Q_i$$

$$P_1^i = P_0^i - Q$$

Then calculate new Instantaneous Utilization factor for i^{th} task by using formula 1 and 2-

$$U_1^i = C_1^i / P_1^i$$

$$U_1^1 = C_0^1 - Q_1 / P_0^1 - Q = 3-0/9-1 = 0.37$$

$$U_1^2 = C_0^2 - Q_2 / P_0^2 - Q = 5-1/11-1 = 0.40$$

$$U_1^3 = C_0^3 - Q_3 / P_0^3 - Q = 7-0/38-1 = 0.18$$

Now we get the new task set as

TABLE II. TASK SET AFTER ONE QUANTUM OF EXECUTION.

T_i	C_i	P_i	U_i
T_1	3	8	0.37
T_2	4	10	0.40
T_3	7	37	0.18

3. Again here we observed that the task T_2 has higher instantaneous utilization so allow it for execution and recalculate the task set for the next quantum of execution time.

C_1^1=3 P_1^1=8 Q_1=0

C_1^2=4 P_1^2=10 Q_2=1

C_1^3=7 P_1^3=37 Q_3=0

$Q=\sum (Q_i) =0+1+0=1$

Now using the formula,

$$U_2^i = C_1^i - Q_i / P_1^i - Q$$

$$U_2^1 = C_1^1 - Q_1 / P_1^1 - Q = 3-0/8-1 = 0.42$$

$$U_2^2 = C_1^2 - Q_2 / P_1^2 - Q = 4-1/10-1 = 0.33$$

$$U_2^3 = C_1^3 - Q_3 / P_1^3 - Q = 7-0/37-1 = 0.19$$

Now we get the new task set as:

TABLE III. TASK SET AFTER SECOND QUANTUM OF EXECUTION**.**

T_i	C_i	P_i	U_i
T_1	3	7	0.42
T_2	3	9	0.33
T_3	7	36	0.19

4. Again task T_1 has higher instantaneous utilization so allow it execution and recalculate the task set for the next quantum of execution time .

5. Continue this process till, we get i^{th} quantum of execution time i.e. i=PTC_END_POINT using formula:

$$U_j^i = C_{j-1}^i - Q_i / P_{j-1}^i - Q$$

Now consider run time queue of size PTC and shuffle tasks in run time queue. After shuffling tasks in run time queue, tasks are applied to CPU as shown in fig.3.

D.RESULT ANALYSIS

Given task set is simulated using CHEDDER a real time simulator for RM , EDF and LLF and observed context switching, number of preemptions and deadline missing possibility for each scheduler. We have also designed simulator for our algorithm in C and observed the same parameters with our simulator we are getting following results.

TABLE IV .COMPARISON OF IUF WITH OTHER SCHEDULERS.

Factors	IUF	RM	EDF	LLF
Context Switching	16	13	12	19
CS Ratio	0.42	0.34	0.31	0.50
Response time	Average	High	High	Average
Schedulability	High	Low	High	Average
No. of Preemptions	0	3	4	9

It is clear from fig. 4 and 5 that, although context switching is high comparative to other algorithms but number of preemptions are zero hence it increases schedulability of tasks .

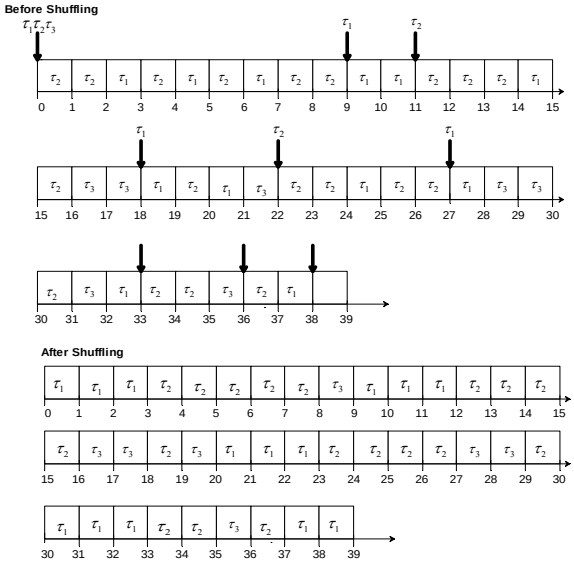

Figure 3.Shuffling Of Tasks Using Run Time Queue For A Given Tasks Set.

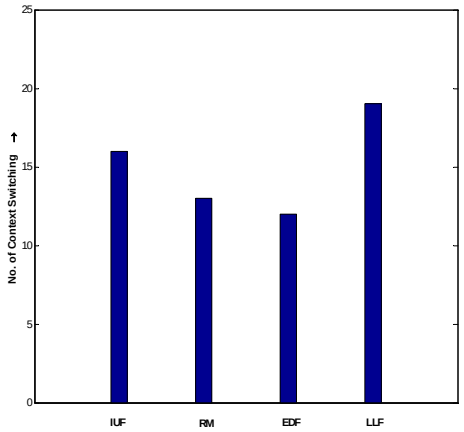

Figure 4: Comparative context switching of IUF with other Scheduling algorithms

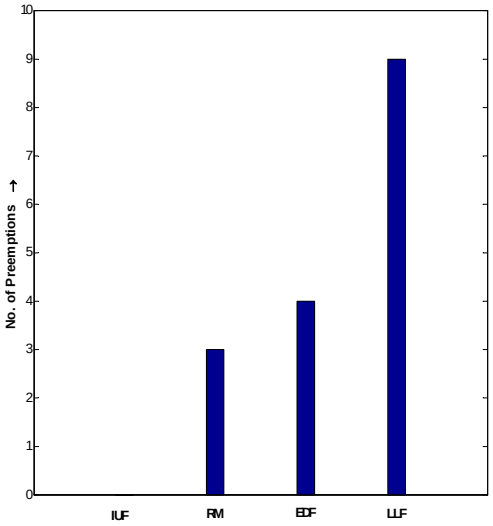

Figure 5: Comparative preemptions of IUF with other Scheduling algorithms

IUF scheduling algorithm [14] has the drawback that context switching is very high since in this algorithm, shuffling of tasks is suggested; this loses intention of priority assignment.
Therefore there is need to modify IUF scheduling algorithm so as to reduce context switching. In proposed framework tasks are logically divided into mandatory and optional portion. All mandatory portions are scheduled according to instantaneous utilization like IUF. This paper proposes a new approach where imprecise computation model is used ,where task is logically divided into two parts, mandatory and optional part. Priority of the mandatory portion of task is based on this IUF. Since the IUF is a temporally variant factor, the priority of each task will vary continuously. Optional portions are scheduled by shortest job first. Experimentally it is proved that it increases CPU utilization effectively. It also handles harmonic and non harmonic periods comparably. Number of missing deadlines is less. In order to improve reliability, the active mandatory portion is loaded as redundant copy so that if at all mandatory portion is failed due to some reason, at least mandatory portion will get executed. Thus by compromising in performability, it enhances reliability through redundancy. The main advantage of this framework is that there is no need to include error recovery points in program which avoids system overheads.

IV. MODIFIED IUF REAL TIME SCHEDULING ALGORITHM

A block diagram of proposed framework is shown in the figure 1. This is basically a planning based offline preemptive scheduling algorithm where priorities are assigned based on

Instantaneous Utilization. Every task is scheduled for fixed quantum of time i.e. for mandatory part of task.

Basic assumptions of the system

1. Given task set is considered as IRIS task (Increased reward with increased service).
2. Initially task is divided into two parts:
   i) Mandatory portion,
   ii) Optional portion.
3. Consider arrival time of all task =0.
4. Period of task is equal to its relative deadline.
5. The soft real-time system is considered where system will tolerate lateness with decreased service quality but no critical consequences.

B. System Model

Phase 1: Divide the task into two parts

Consider two portion of the task from the given task set as
i) Mandatory portion, ii) Optional portion.

Phase 2: scheduling of the mandatory portion of task using IUF

Step 1:- Initially for given task set, calculate CPU utilization of each task using formula

$$U_0^i = C_0^i / P_0^i \qquad (8)$$

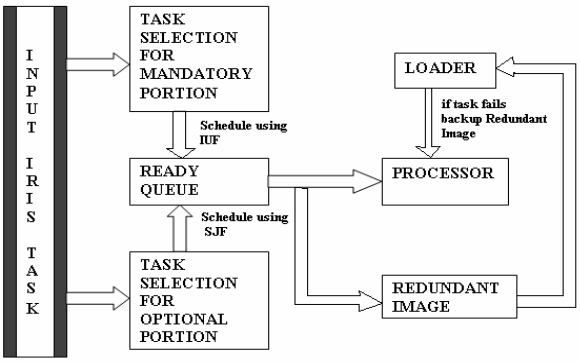

Figure 6: Architecture of MIUF Framework.

U_0^i =Initial utilization of i th task.

C_0^i =Initial computation time of.

P_0^i =Initial period of invocation.

Based on utilization [U_0^i] the task which is having higher value of Utilization, mandatory portion of this task is mapped to the CPU.

Step 2:- One task has executed for one quantum (equal to mandatory portion) of time. Again calculate value of $\sum Q_i = Q$, P_1^i by using following formula-

$$C_1^i = C_0^i - Q_i . \qquad (9)$$

$$P_1^i = P_0^i - Q . \qquad (10)$$

Where $\sum Q_i = Q$,

Then calculate new Instantaneous Utilization factor for ith task for using formula 1 and 2

$$U_1^i = C_1^i / P_1^i \qquad (11)$$

Where,

C_1^i =Instantaneous computation time for ith task

P_1^i = Instantaneous period of execution of ith task.

U_1^i =Instantaneous Utilization of ith task.

For second iteration of time derive the table ($T_i, C_1^i , P_1^i , U_1^i$)

Again the task which is having highest instantaneous utilization will be having highest priority of execution for second iteration quantum.

Like wise, calculate

$$C_j^i = C_{j-1}^i - Q_i \qquad (12)$$

$$P_j^i = P_{j-1}^i - Q \qquad (13)$$

Calculate U_j^i using equation 4 and 5

$$U_j^i = C_j^i / P_j^i \qquad (14)$$

Where,

U_j^i =Instantaneous utilization of ith task for the jth iteration of quantum.

At any given instant of time, If it is observed that, instantaneous utilization is greater than U_0^i , it means that task is going to miss its deadline.

Phase 3: scheduling the optional portion of the task

For scheduling the optional portion, shortest optional portion first is employed.

ALGORITHM

1) Take input of tasks containing period, mandatory execution time, and optional execution time.
2) Calculate the mandatory utilization and optional utilization for each task.

3) Check the schedulability for tasks according to their utilization
4) Generate CPU mapping for tasks.
5) Execute the mandatory portion of tasks according to the highest instantaneous utilization. Meanwhile if there is interrupt due to corruption of task while executing the mandatory portion of task then a backup image is maintained so that the task can be restored from the image.
6) After executing mandatory portion of all tasks execute optional portion according the shortest job first policy.
7) If interrupt occurs due to corruption of task in the optional portion of task then optional portion does not run to its completion and gets aborted.

B. Result Analysis

Various tasks sets have been applied to MIUF scheduler simulator and it has been observed that context switching, response time and CPU utilization is improved .Following table I to X shows how tasks are selected and scheduled using MIUF.
Here, M= Mandatory portion of task for execution,
O= Optional Portion of the task for execution,
P = Period/Deadline of task.

TABLE V: GIVEN TASK SET WITH EXECUTION TIME AND PERIOD

Task	M	O	P
T1	2	2	18
T2	3	2	20
T3	2	1	16
T4	2	1	15

TABLE VI: SELECTION OF FIRST MANDATORY TASK

Task	M	O	P	U(M)
T1	2	2	15	0.13
T2	0	2	17	0.00
T3	2	1	13	0.15
T4	**2**	**1**	**12**	**0.16**

TABLE VII: SELECTION OF SECOND MANDATORY TASK

Task	M	O	P	U(M)
T1	2	2	18	0.11
T2	**3**	**2**	**20**	**0.15**
T3	2	1	16	0.12
T4	2	1	15	0.13

TABLE VIII: SELECTION OF THIRD MANDATORY TASK

Task	M	O	P	U(M)
T1	2	2	13	0.15
T2	0	2	15	0.00
T3	**2**	**1**	**11**	**0.18**
T4	0	1	10	0.00

TABLE IX: SELECTION OF FOURTH MANDATORY TASK

Task	M	O	P	U(M)
T1	**2**	**2**	**11**	**0.18**
T2	0	2	13	0.00
T3	0	1	9	0.00
T4	0	1	8	0.00

TABLE X: SELECTION OF FIRST OPTIONAL TASK

Task	M	O	P	U(M)
T1	0	2	9	-
T2	0	2	11	-
T3	0	1	7	-
T4	**0**	**1**	**6**	-

TABLE XI: SELECTION OF SECOND OPTIONAL TASK

Task	M	O	P	U(M)
T1	0	2	8	-
T2	0	2	10	-
T3	**0**	**1**	**6**	-
T4	0	0	5	-

TABLE XII: SELECTION OF THIRD OPTIONAL TASK

Task	M	O	P	U(M)
T1	**0**	**2**	**7**	-
T2	0	2	9	-
T3	0	0	5	-
T4	0	0	4	-

TABLE XIII: SELECTION OF FOURTH OPTIONAL TASK

Task	M	O	P	U(M)
T1	0	0	5	-
T2	**0**	**2**	**7**	-
T3	0	0	3	-
T4	0	0	2	-

TABLE XIV: FINALLY ALL TASKS ARE GETTING SCHEDULED

Task	M	O	P	U(M)
T1	0	0	3	-
T2	0	0	5	-
T3	0	0	1	-
T4	0	0	0	-

Here, the table XV indicates the complete case study for the given set of tasks. It also indicates the time interval and the description how the task get schedules.

TABLE XV: DESCRIPTION OF TASK SCHEDULE.

Time Interval	Executing Task	Description
0	M(T2)	Mandatory part of Task T2 has highest instantaneous utilization so it gets executed first until its computation is over.
3	M(T4)	Now, Mandatory part of Task T4 has highest instantaneous utilization so it gets executed.
5	M(T3)	Then, Mandatory part of Task T3 has highest instantaneous utilization so it gets executed.
7	M(T1)	Finally, Mandatory part of Task T3 has highest instantaneous utilization so it gets executed.
9	O(T4)	After completing mandatory part of all tasks, Task T3 has lowest optional part so it gets executed. Here optional portion are equal tie is broken on FCFS.
10	O(T3)	Now, Task T4 has lowest Optional part so it gets executed.
11	O(T1)	Then, Task T1 has lowest optional part so it gets executed.
13-15	O(T2)	At last, Task T2 has lowest optional part so it gets executed.
-	-	All tasks have completed their mandatory as well as optional portion. Second iteration starts.

The said algorithm is simulated in Microsoft Visual Basic 6 and existing algorithms are simulated using CHEDDER simulator.

1. We attempt to keep the framework as general as possible by accommodating 'software' faults tolerated by some form of recovery block, and 'hardware' faults dealt with by state restoration and re-execution. Error latencies will be assumed to be short and hence tried to enhance reliability of scheduler.
2. The previous IUF algorithm has more number of context switches as the quantum considered is too small. But in the proposed algorithm, the context switches get minimized as the quantum considered is the mandatory portion of task(fig.7) The task switch from one to another at every unit instance. But for proposed algorithm it is very less. The following table shows the analysis of the results for previous IUF algorithm and proposed algorithm.

TABLE XVI: COMPARATIVE RESULTS OF VARIOUS CASE STUDIES

	Context switching		Response time (T1, T2, T3)		CPU Utilization (In %)	
Case study	IUF	MIUF	IUF	MIUF	IUF	MIUF
1	15	7	14,15,12	10,11, 13	17	22
2	11	5	12 ,11, 10	7, 9 ,10	22	28
3	10	5	11,12,10	8,10,12	20	25

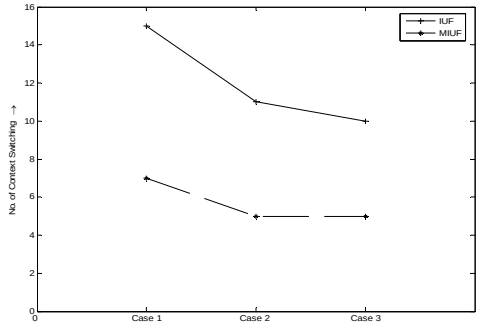

Figure 7: Comparative Context Switching of MIUF and IUF

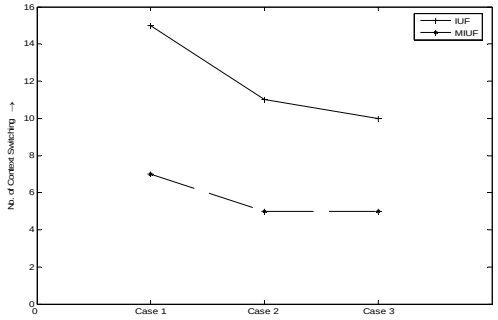

Figure 8: Comparative schedulability of MIUF and IUF scheduling algorithm.

Schedulability of MIUF is improved than IUF scheduling algorithm.(Fig.8).

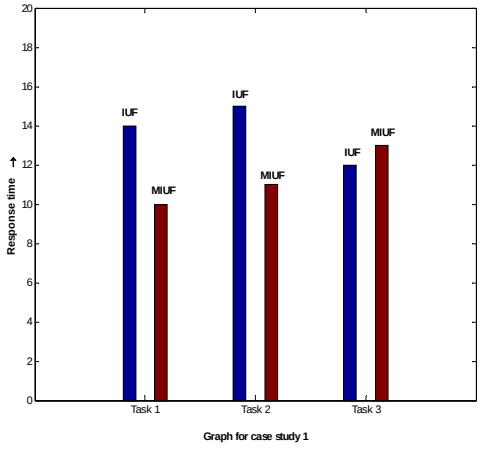

Figure 9: Comparative response time of MIUF and IUF scheduling algorithm.

Response Time observed in MIUF is less than that of IUF(fig.9) also in MIUF CPU utilization is also increased considerably.

V. CONCLUSION

Approach of IUF is better than traditional scheduling approaches. First attempt of IUF has better performance in terms of number of preemptions and schedulability.Second attempt does better than IUF in context switching, response time and schedulability.

In order to reduce context switching observed in IUF, an approach of IRIS is used i.e. task is logically divided into two parts i) Mandatory portion and ii) Optional portion. With scheduling mandatory portion by highest instantaneous utilization first and optional portion by shortest optional portion first, results observed are encouraging. It has been observed that context switching, response time and CPU utilization has been increased as compared with IUF scheduling algorithm. This framework also suggests redundancy to mandatory portions so as to increase reliability of schedulers. Introducing this concept, the necessity of adding error checking bits is removed. It increases resource utilization by compromising in the performability.

REFERENCES

[1] C. L. Liu and J. W. Layland,"Scheduling algorithms for multiprogramming in a hard-real-time environment,"*Journal of the ACM*, 20(1), pp.47-61January 1973.

[2] Leung J. and Merrill M., A note on preemptive scheduling of periodic real-time tasks, *Information Processing Letters*, **11**(3): 115–118, 1980.

[3] Dhall S.K., Scheduling periodic-time critical jobs on single processor and multiprocessor computing systems, PhD thesis, University of Illinois, April 1977.

[4] Sorenson P.G., A methodology for real-time system development, PhD Thesis, University of Toronto, Canada, 1974.

[5] D. B. Stewart and P. K. Khosla, "Real-Time Scheduling of Sensor-Based Control Systems," in *Real-Time Programming* (W. Halang and K. Ramamritham, eds.), Tarrytown, NY: Pergamon Press, 1992.

[6] K.Ramamritham,J.A. Stankovic, P.Shiah, "Efficient scheduling algorithms for real time multiprocessor system," IEEE Transaction on parallel and distributed system, 1(2):pp.184-94. Apr, 1990.

[7] J. Goossens, S. Funk, and S. K. Baruah, "Priority-Driven Scheduling of Periodic Task Systems on Multiprocessors." Real-Time Systems, vol. 25, no. 2-3, 2003, , pp. 187-205.

[8] T. P. Baker, "Multiprocessor EDF and Deadline Monotonic Schedulability Analysis." in IEEE Real-Time Systems Symposium (RTSS), 2003, pp. 120129.

[9] K.J. Lin, S. Natarajan, and J. W. S. Liu, Concord ," A distributed system making use of imprecise results" in *Proc. of COMPSAC '87*, Tokyo, Japan, 1987.

[10] X. Castillo, S.P. McConnel, and D.P. Siewiorek. "Derivation and Calibration of a Transient Error Reliability Model,"*IEEE Transactions on Computers*,31(7), July 1982,pp. 658–671.

[11] H. Kim, A.L.White, and K. G.Shin " Reliability modeling of hard real-time systems ," In *Proceedings 28th Int. Symp. on Fault-Tolerant Computing (FTCS-28),*. IEEE Computer Society Press, 1998, pp 304–313.

[12] A. Campbell, P. McDonald, and K. Ray. Single "Event upset rates in space" *IEEE Transactions on Nuclear Science*, 39(6): December 1992, pp.1828–1835.

[13] Arshad Iqbal,Asia Zafar,Bushra Siddique,"Dynamic Queue Deadline first scheduling algorithm for soft real-time System",IEEE International Conference on Emerging Technologies, sept. 17-18 Islamabad,2005, pp. 346-351.

[14] Radhakrishna Naik, Vivek Joshi, R. R. Manthalkar, "IUF Scheduling Algorithm for Improving the Schedulability, Predictability and Sustainability of the Real Time System," ICETET-09,IEEE Second International Conference on Emerging Trends in Engineering & Technology, pp.998-1003, 2009.

[15] Radhakrishna Naik, R. R. Manthalkar, "Modified IUF Scheduling Algorithm for the Real Time Systems," ICETET-10,IEEE third International Conference on Emerging Trends in Engineering & Technology, pp.712-716, 2010.