ISSN (Online): 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

Survey on Optimal Schedulability Test for Hard Real Time Scheduling

Nitin C. Patil and Prof. S. P. Dhanure
Dept. of Electronics and Telecommunication Engg, SITS, Narhe, Pune
Savitribai Phule Pune University, Pune, Maharashtra, India.

Abstract : Real time system is now ubiquitous. In such real time system requires the exactness in result of logical behavior and physical occurrence at which the results are produced. Throughout the paper we consider only hard real time tasks implement on uniprocessor and if dead line is missed in such system then its ruinous. Many scheduler had been proposed to improve the schedulability, here we talks about dynamic and static scheduling only like Earliest Deadline First (EDF) and Rate Monotonic (RM) and considering a scenario of only $D \leq T$. We analyze tasks in every condition to check its schedulability with respected solution to each problem. First we check utilization of tasks to verify its schedulability, but for optimal schedulability we required response time analysis. Again during task interaction some issues occur such as blocking, priority inversion and jitter which also need to verify. That all issues where discuss and analyze throughout the paper with the help of examples with their solutions.

Keywords : Real time system, task scheduling, EDF, RM, utilization, response time analysis, priority inversion, jitter.

I. INTRODUCTION

A real-time scheduling System is composed of the scheduler, clock and the processing hardware elements. In a real-time system, a process or task has schedulability; tasks are accepted by a real-time system and completed as specified by the task deadline depending on the characteristic of the scheduling algorithm [1]. Modeling and evaluation of a real-time scheduling system concern is on the analysis of the algorithm capability to meet a process deadline. A deadline is defined as the time required for a task to be processed.

A real-time scheduling algorithm can also be classified as static or dynamic. Tasks are accepted by the hardware elements in a real-time scheduling system from the computing environment and processed in real-time. An output signal indicates the processing status. A task deadline indicates the time set to complete for each task. A task deadline for a static scheduler is predetermined offline. A different alternative is to schedule a task when the system is running; this process is known as Dynamic scheduling[1][2].

It is not always possible to meet the required deadline; hence further verification of the scheduling algorithm must be conducted. Two different models can be implemented using a dynamic scheduling algorithm; a task deadline can be assigned according to the task priority (earliest deadline) or a completion time for each task is assigned by subtracting the processing time from the deadline (least laxity)[1]. Deadlines and the required task execution time must be understood in advance to ensure the effective use of the processing elements execution times.

While designing the scheduling of any task, we should consider some notations like deadline, period & Worst Case Execution Time (WCET)[1][9]. Throughout the paper we'll talk the scheduling test examine on single processor. $Utilization$ is the well known concept design by Liu and Layland in their seminal paper[8]. It is simple way to find the schedulability test of real-time tasks. For static priority say RMS the paper [8] design the utilization bound test whereas for dynamic priority it required only simple utilization test.

If we calculate $utilization$ then its not enough to check its schedulability whether it may satisfy the condition, where different conditions of shedulability are their for different scheduleing algorithms[5]. If utilization test satisfy then go for Workload[7] where it checks the processing load of all tasks handle by CPU or not. Again the largest interval L upto where we want to check the processor load that can be calculated by using processor Demand Bound function[10][11]. But the point where the processor capacity overloaded then such point we called as Buruah Point which tell the maximum processing capacity we'll be hold by scheduler. Thus it is all about the boundary of task scheduling.

For optimal Scheduling, Response Time Analysis[7][12] is required to calculate, which is compare with the respected task deadline and if the response time is less than its deadline then the task is not schedulable. A very simple analysis but again many complex scenario comes during execution which has to consider for optimal scheduling. During task interaction like preemption or sharing resources some mischievous behavior seen like Priority Inversion which can be prevented by using priority ceilings or priority boosting. And in some scenario a lower priority task block the higher priority task causing blocking of hard real time tasks. Also because of poor system performance, some delay occurs in between invocation and execution of task called as Jitter may causing misses deadline of low priority task, which is difficult to predict[16][17].

The whole scenario we'll discuss in the paper using examples considering every parameter which affecting the real time task

DOI 10.17148/IJARCCE.2015.45158

ISSN (Online): 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

scheduling.

The paper flow is given as: Section-II presents literature survey of paper. Section-III discuss scheduling algorithms that we going to analyze in the paper. Section-IV discuss an example to calculate the response of task with the help of algorithm. Section V discuss the general issues occur during task interaction like blocking of task, priority inversion, cooperative scheduling to reduce WCET in some extend and analysis of jitter. Section VI conclude the paper.

II. Literature Survey

We conducted survey on Real Time System (RTS) and then its related parameters which affect the hard real time scheduling and find solution over such unpredictable issues and then we'll come up with some tests to check task set schedulable or not.

A. *Real Time System*

A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or it is useless! Where RTS is divided into two categories: Hard RTS and Soft RTS. A Hard Real-Time System guarantees that critical tasks complete on time. This goal requires that all delays in the system be bounded from the retrieval of the stored data to the time that it takes the operating system to finish any request made of it. A Soft Real Time System where a critical real-time task gets priority over other tasks and retains that priority until it completes. As in hard real time systems kernel delays need to be bounded[1]. A RTS architecture is totally based on hybrid control architecture as shown below fig. 1:

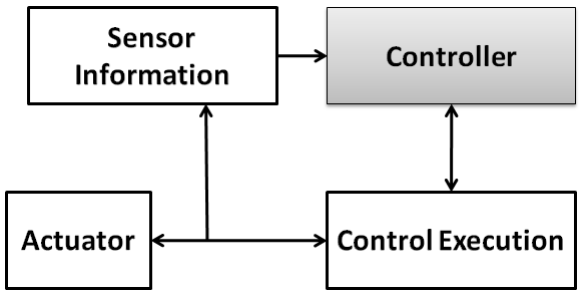

Figure 1: Real Time System Architecture

The first to discuss is periodic tasks. A periodic task is performed on a given frequency. A periodic task is described by $\top_i$, $i \in (1,...,n)$. Every periodic task has its own relative deadline D_i, with minimum period of T_i and maximum computational time C_i can be say as worst case execution time. A Invocation of any task known as a job. The k^{th} invocation of task i is denoted as $\top_i$, $k \in (1,2,...)$. Each invocation of a task has its own absolute release time r_i^k and absolute deadline d_i^r. Constraint for the absolute values of periodic tasks are, $d_i^r = r_i^k + D_i$, $d_i^r \leq r_i^{k+1}$ and $r_i^{k+1} - r_i^k \leq T_i$ for any $i \leq 1$, $k \leq 0$. Each job has its own absolute starting time S_i and absolute ending time E_i. Using absolute ending time and absolute deadline, the time left by task till its deadline can be calculated, known as Lateness L_i. The lateness of job can be stated as $E_i - d_i = L_i$, in general lateness is negative. The tasks which is release at random time called as sporadic tasks. parameter for these tasks are same as periodic tasks only the difference is the constraint used for release time. For sporadic tasks the $(k+1)^{th}$ invocation occurs at $r_i^{k+1} \geq r_i^k + T_i$[1][7] Below fig. 2 shows the contains discuss.

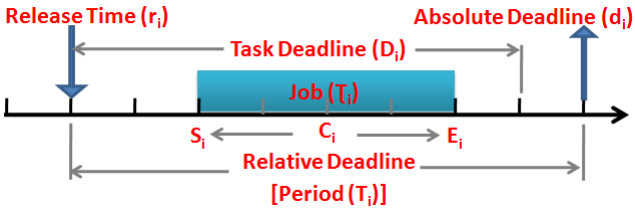

Figure 2: Periodic task model with its timing parameter

Based on above task model below utilization, workload, processor demand and Baruah point will be define.

B. *Utilization:*

a) *Earliest Deadline First:*
The fraction of CPU time spent executing the task set called as Utilization U[5][6], where if

- $U > 1$, then the schedule is not feasible or overload.
- $U \leq 1$, can be schedulable but still depend on scheduling algorithm.
- $U = 1$, kept CPU busy i.e., all deadline will be met.

$$\boldsymbol{x(L)} = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{1}$$

b) *Rate Monotonic:*
The utilization based analysis for Rate Monotonic (RM) is simple sufficient but not necessary schedulability test. $U < 1$ doesn't imply the tasks are schedulable with RM, it required a special test say Utilization Bound Test (UB). The UB test is proposed by Liu and Layland, 1973[6] as shown below. The Utilization should be ≤ 0.69 as $n \to \infty$[5][7].

$$\boldsymbol{x(L)} = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq N(2^{1/N} - 1) \tag{2}$$

C. *Workload:*

The workload on processor by all n tasks, between length 0 to L[7], is define by:

$$\boldsymbol{W(L)} = \sum_{i=1}^{n} \left\lceil \frac{L}{T_i} \right\rceil C_i \tag{3}$$

The function is the release time of i^{th} tasks upto the time t and multiply by computational time of i^{th} task. The workload is the summation of function of all tasks.

ISSN (Online): 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

D. *Processor Demand:*

We can calculate N_i^L by counting how many times task $\top_i$ has arrived during interval [*0, $L - D_i$*]. We can ignore instance of the task that arrived during the interval $[L - D_i, L]$ for these instances. where the largest interval, L is the LCM of the task's periods and only absolute deadline need to be consider[7]. We can express N_i^L as shown in fig. 3

$$N_i^L = \frac{L - D_i + T_i}{T_i} \tag{4}$$

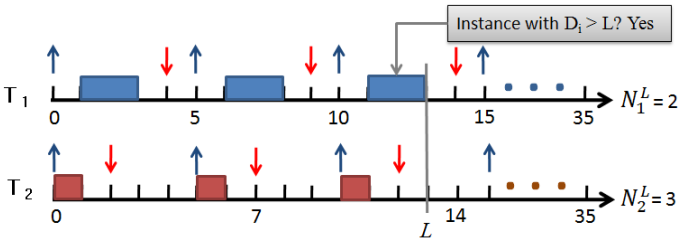

Figure 3: Example to calculate N_i^L

The load that should be adjudicated by the processor, between $L = 0$ and $L = n$, is define by:

$$H_p(0, L) = \sum_{i=1}^{n} N_i^L \cdot C_i \tag{5}$$

The sufficient and necessary condition for EDF scheduling for which $D_i \leq T_i$, is

$$\forall \boldsymbol{L} : H_p(0, L) \leq L \tag{6}$$

where $H_p(0, L)$ is the total processor demand in *[0, L]*.The processor-demand analysis and associated feasibility test was presented by S. Baruah, L. Rosier and R. Howell in 1990 [10]. Baruah [10] derived an upper bound for the processor demand function. Using this upper bound we can find the line crosses the processor capacity, $t = L_B$ line. After this point processor cannot manage the tasks load.

E. *Baruah Point:*

Baruah Point is the limit, say L_B, of processor capacity, and processor demand is get lower down after the L_B line[10]. L_B is given as:

$$L_B = \frac{\sum_{i=1}^{n} \left(1 - \frac{D_i}{T_i}\right) \cdot C_i}{1 - U} \tag{7}$$

Assumptions Some assumption need to be used are as follows,

- All tasks are periodic in nature.
- The scheduler used algorithm with highest priority
- Scheduling overhead are assumed to be included in computation time C_i of task.
- All tasks are independent of each other.
- Each task must be complete before next request occur.

F. *Response-Time Analysis:*

Mathai Joseph and Paritosh Pandya[12], 1986 proposed a simple idea for sufficient and necessary schedulability analysis. The worst case response time for all tasks is given when all tasks are released at the same time called as critical instance. Calculate the worst case response time *R* for each task with deadline *D*. If $R_i \leq D_i$ or $R_i = C_i + I_i$ the task is schedulable/feasible, where *I* is interference from higher priority tasks. Repeat the same check for all tasks. If all tasks pass the test, the task set is schedulable. Now the question is how to calculate the worst case response times?. During R, each higher priority task j will execute a number of times: Number of Release = $\left\lceil \frac{R_i}{T_j} \right\rceil$. The ceiling function $\lceil \ \rceil$ gives the smallest integer greater than the fractional number on which it acts. So the ceiling of 1/3 is 1, of 6/5 is 2, and of 6/3 is 2. The total interference is given by, $\left\lceil \frac{R_i}{T_j} \right\rceil C_i$[7].

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{8}$$

Where *hp(i)* is the set of tasks with priority higher than task *i*. Now solve the above equation using recurrence relationship.

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \in hp(i) \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j \tag{9}$$

The set of values $w_i^0, w_i^1, w_i^2, ..., w_i^n$, is monotonically non decreasing. When $w_i^n = w_i^{n+1}$ the solution to the equation has been found, w_i^0 must not be greater than R_i (e.g. 0 0r C_i)[7].

G. *Priority Inversion and Blocking:*

Only the response time analysis is not sufficient to pass the test of being task is schedulable or not. We have to consider some issues occur during task interaction say priority inversion. Here we discuss an example to clarify the concept of priority inversion, whatever the solution that we'll discuss detail in section V. Imagine a system with three active threads as shown in fig. 4. The task serial number consider as the respected priorities and assume their is no other tasks of higher priority in the system.

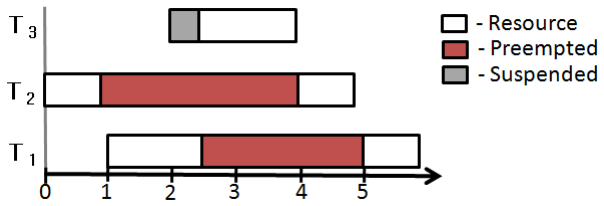

Figure 4: Example for Priority Inversion

When the system begins execution, thread $\top_3$ is released and executes immediately since there are no other higher priority threads executing. Shortly after it starts, it acquires a lock on resource R shown by white color in fig. 4. At time $t = 1$, thread $\top_1$ is released and preempts thread $\top_2$ since it's of higher priority, the red color shows the preempted part. At

ISSN (Online): 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

time $t = 2$, thread $\top_3$, a medium priority thread, is released but doesn't execute because higher priority thread $\top_1$ is still executing. Shortly afterward, however, thread $\top_1$ attempts to acquire a lock on resource R, but cannot since thread $\top_2$ (a lower priority thread) still owns it. This allows thread $\top_3$ to execute in its place, which effectively violates the priority-order execution of the system, resulting in what we call priority inversion[13].

In this situation, thread $\top_1$ will continue to block on resource R, for an unbounded and unknown amount of time, until thread $\top_3$ blocks (or terminates). For a real-time system, where thread $\top_1$ controls something time-critical (i.e. the ailerons on an airplane to maintain level flight), the result can be disastrous. When $\top_3$ finally does block, thread $\top_2$ will continue execution and release its lock on R. At that point, thread $\top_1$ will preempt it, acquire its lock on R, and continue. But it may be too late, and execution deadlines may have been missed. Let us see some analysis

- If the system has k critical sections that can lead to a task $\top_i$ being blocked then the maximum number of times that $\top_i$ can be blocked is k.
- If B is the maximum blocking time and k is the number of critical sections, the process i has an upper bound on its blocking given by[13][14]

$$B_i = \sum_{k=1}^{k} use(k,i)C(k) \tag{10}$$

If we incorporate the blocking B in response time R then the Eq. 8 & Eq. 9[7][10] becomes,

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{11}$$

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \in hp(i) \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j \tag{12}$$

H. *Jitter*

Jitter is the difference between succeeding periods of time for a given task. The difference in invocation (arrival) time and its release time (start execution) of same task leads to jitter (delay). Jitter may have different cause but jitter does not preempt or not get preempted by tasks[17], since it is not a task to do so. If we added the jitter in our basic Eq. 8, then the equation becomes,

$$R_i = C_i + J_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{13}$$

Consider a task i, so the J_i^{max} is maximum jitter and J_i^{min} is minimum jitter of task. The the jitter can be define as,

$$J_i = J_i^{max} - J_i^{min} \tag{14}$$

Where the lower priority task can be preempted twice because of jitter if $R_L > J_i$, so we can say that two preemption of task L occur if $R_L > T_H - J_H$, thrice if $R_L > 2T_H - J_H$ and four times if $R_L > 3T_H - J_H$ and so on. Then the L can be preempted n times if $R_L > (n-1)T_H - J_H$, then, $\frac{R_L + J_H}{T_H} > n - 1$. The largest value of n given by the ceiling function is, $n = \left\lceil \frac{R_L + J_H}{T_H} \right\rceil$[17]. So if we modify our response time Eq. 11 then the equation to be,

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_L + J_H}{T_H} \right\rceil C_j \tag{15}$$

Now how this response time is calculated ? it required to calculate WCET (w_i)[9], so it is given as,

$$w_i^{n+1} = C_i + B_{max} + \sum_{j \in hp(i)} \in hp(i) \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j \tag{16}$$

If the task is release at time t, then in worst case scenario the higher priority tasks can interface between $t + J_i^{max}$ and $t + R_i$, so the worst case response time is given by, $R_i = J_i^{max} + w_i$. In case of $D_i \leq T_i$ the tasks are not optimal to schedule with jitter. We have to order the priority on $D_i - J_i^{max}$ so that the maximum jitter leads to cause task less deadline and acquire the higher priority[17].

III. Scheduling Algorithm

The discuss scheduling algorithms are as follows,

- Earliest Deadline First
- Rate Monotonic Scheduling

A. *Earliest Deadline First*

The *Earliest Deadline First* scheduling algorithm, presented in [1], works with the following rules:

- Each task priority is a function of its period.
- As time proceed and task deadline comes closer, its priority increases proportionally i.e., dynamic in nature.
- Highest priority task is allowed to run first.

Consider the set of independent periodic tasks, but it is not necessary because EDF can works for all periodic as well aperiodic types of tasks. Whenever a new task is arrive, it sorts the ready queue so that the task closest to the end of its period assigned the highest priority. Generally EDF is optimal i.e. EDF can schedule any task set if any one else can[4].

Example 1: Consider two periodic tasks, $\top_1$ and $\top_2$ having task set in terms of (C_i, T_i) and $D_i = T_i$. So the task sets are *(2, 5), (4, 7)*. Then according to Eq. 1 Utilization U of two periodic task calculated as,

$$\boldsymbol{U = \frac{2}{5} + \frac{4}{7} \cong 0.97} \tag{17}$$

As $U < 1$, we can say that task is schedulable. If $D_i \leq T_i$, i.e., $D_1 = 4$ *and* $D_2 = 6$ then only $U < 1$ is not sufficient to tell the schedulability, we required exact feasibility test to check the schedulability. Consider $L = 14$, from Eq. 4 $N_1^{14} = 3$ and $N_2^{14} = 2$. Now calculate processor demand from Eq. 5[1][5][4]

$$H(0, L) = 3 \times 2 + 2 \times 4 = 10 \tag{18}$$

So from Eq. 17 & Eq. 6 ($10 \leq 14$), we can say that the tasks are schedulable by EDF.

ISSN (Online): 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

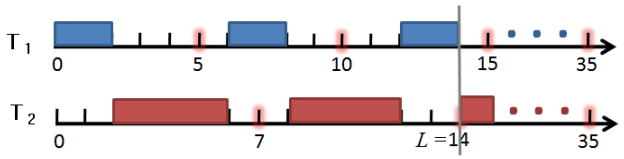

Figure 5: Example for EDF scheduling having two periodic task set

Summary:

- It works with all tasks periodic as well aperiodic.
- It has simple schedulability test: $U \leq 1$
- It is optimal and gives best CPU utilization.
- **Difficult to Implement:** In practice, It is not very often adopted due to the dynamic priority-assignment (expensive to sort the ready queue on-line), which has nothing to do with the periods of tasks. Note that Any task could get the highest priority.
- **Non Stable:** If any task instance fails to meet its deadline, the system is not predictable, any instance of any task may fail

B. Rate Monotonic

The Rate Monotonic scheduling algorithm is works on the following assumption:

- The priority of a task is a monotonically decreasing function of its period.
- Priority of all tasks is static in nature, task with smaller period get higher priorities.
- Tasks are independent and always released at the start of their period.

Consider the set of independent periodic tasks. Here as like EDF, RM doesn't deal with aperiodic types of tasks. So we can not consider a new task to be arrived. According to priority the scheduler sort task, and allow highest priority task to run first every time[1][4].

Example 2: Consider the same task set as we seen in *example 1*. According to utilization of $\backsim$0.97, using Rate Monotonic scheduling algorithm the tasks are schedulable or not ?; let us see in below example and then analyze the conclusion. Here the task having less period gets highest priority i.e., Priority($\top_1$)=1 and Priority($\top_2$)=2. N is depend on he number of tasks, here $N = 2$.

$$N(2^{1/N} - 1) = 2(2^1 - 1) = 82.8 \tag{19}$$

From Eq. 17 and Eq. 19,

$$\sum_{i=1}^{n} U_i \nleq N(2^{1/N} - 1) \tag{20}$$

So from Eq. 2 and Eq. 20, we can say that the tasks are not schedulable by Rate Monotonic scheduling algorithm as clearly shown in fig. 7. From the graph below of fig. 6 it is easy to analyze that the tasks are schedulable or not comparing to utilization.If we draw the graph as shown below for the test of schedulability then it is easy to analyze the schedulability condition for Rate Monotonic scheduling algorithm. IF the Utilization U from Eq. 1 is above the red line of graph of fig. 6, then the tasks of real time system under RM is not schedulable and vice versa. In fig. 7 it is clearly seen that the task are not schedulable under RM because it misses the second task deadline.

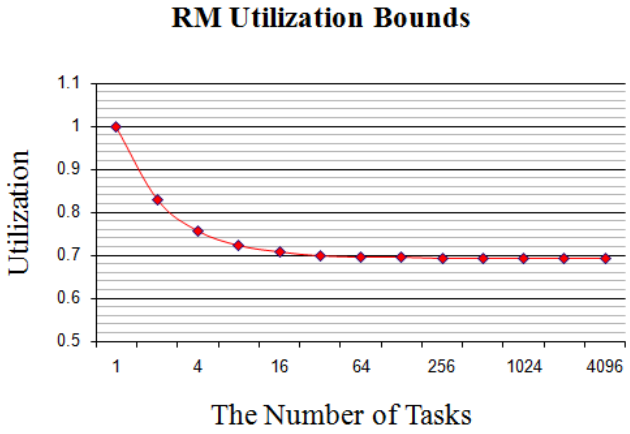

Figure 6: Real-time system is schedulable under RM if Eq. 2 satisfy

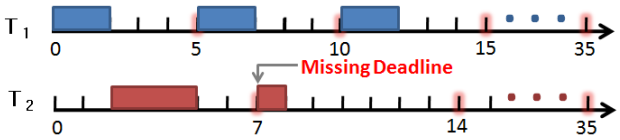

Figure 7: Example for RM Scheduling having two periodic task set

Summary:

- It works with only periodic task not aperiodic tasks.
- Because of static priority, expenses to sort queue in run time is reduces i.e., Ease to implement.
- Simply schedulability test is not sufficient to tell the task are schedulable or not. So additionally *Utilization Unbound Test* required to check the schedulability.

Note that in our examples, we have assumed that all tasks are released at the same time: this is to consider the critical instant (the worst case scenario). If tasks meet the first deadlines (the first periods), they will do so in the future (why?). Critical instant of a task is the time at which the release of the task will yield the largest response time. It occurs when the task is released simultaneously with higher priority tasks. Note that the start of a task period is not necessarily the same as any of the other periods: but the delay between two releases should be equal to the constant period (otherwise we have jitters). This above observation we will see in the next section in detail.

IV. Response-Time Analysis

As we discuss the calculation of Response-Time in the literature survey, here we analyze the Response-Time[7][12] using a Response-Time Algorithm as shown below.

Let us analyze the above algorithm, consider the same example as we seen before *(2, 5), (4, 7)*. Now follow the below steps using table I below,

ISSN (Online): 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

Algorithm 1 Response Time Algorithm

Require: $n = 0$
Ensure: N is the total no. of jobs
procedure $Start$
  $w_i^n = C_i$
  for $(k = 1; k \leq N; k++)$ **do**
    calculate new w_i^{n+1}
    if $(w_i^{n+1} == w_i^n)$ **then**
      $R_i = w_i^n$
      $Status \leftarrow True$
      $break$
    else if $(w_i^{n+1} > T_i)$ **then**
      $Status \leftarrow False$
    end if
  end for
  if $(Status == True \;\; \&\& \;\; R_i \leq d_i)$ **then**
    return $Schedulable$
  else
    return $Unschedulable$
  end if
end procedure

TABLE I: Tasks parameters values

Period (T)	Computation Time (C)	Priority (P)	
7	4	2	i
5	2	1	j

i) $w_1^0 = 4$
ii) $w_1^1 = 4 + \left(\frac{4}{5}\right) 2 \approx 6$, from Eq. 9
iii) $w_1^2 = 4 + \left(\frac{6}{5}\right) 2 \approx 7$
iv) $w_1^3 = 4 + \left(\frac{7}{5}\right) 2 \approx 7$
v) From step iii & iv, $w_1^2 = w_1^3$, here we are at 5^{th} step of algorithm.
vi) Now in 6^{th} step of algorithm we get the response time of above task, i.e., $R_1 = 7$ and the 13^{th} step shows that the tasks are Schedulable only if $R_i \leq D_i$ and Unschedulable if not satisfied.
vii) So in the above example if, $D_i = T_i$ then the task is Schedulable but if $D_i \leq T_i$ the tasks are Uschedulable

Response Time Analysis[12] is sufficient and necessary. If the process set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a process will miss its deadline (unless the computation time estimations themselves turn out to be pessimistic). During response time analysis the test should obtain the measurement and analysis of Worst-Case Execution Time (WCET) otherwise their may be unpredictable results but the problem with measurement is that it is difficult to be sure when the worst case has been observed. The stumbling block of analysis is that an effective model of the processor (including caches, pipelines, memory wait states and so on) must be available. For such analysis we should need the semantic information. The scope of WCET is not reveal here, for detail information refer[9]. In the next section we'll discuss more parameter which has to consider during test of real time task scheduling.

V. Task Interactions and Blocking

In this section we'll discuss the Priority Inversion occur during resource sharing, where the higher priority task to be suspended by lower priority tasks with unpredictable time. And when the task is waiting for a lower-priority task, it is said to be blocked.

A. *Priority Inversion*

The priority-based model of execution states that a task can only be preempted by another task of higher priority. However, scenarios can arise where a lower priority task may indirectly preempt a higher priority task, in a sense inverting the priorities of the associated tasks, and violating the priority-based ordering of execution. This is called "Priority Inversion", and usually occurs when resource sharing is involved.

The rover failure occur on MARS[15] only because of priority inversion, the failure turned out to be a case of priority inversion. The higher priority bc_dist task was blocked by the much lower priority ASI/MET task that was holding a shared resource. The ASI/MET task had acquired this resource and then been preempted by several of the medium priority tasks. When the bc_sched task was activated, to setup the transactions for the next 1553 bus cycle, it detected that the bc_dist task had not completed its execution. The resource that caused this problem was a mutual exclusion semaphore used within the select() mechanism to control access to the list of file descriptors that the select() mechanism was to wait on.
As we discuss thoroughly in the literature survey with the example, here we find some solutions to eliminate priority inversion[13].

There are a few solutions to the priority-inversion problem in real-time systems. One is to turn off all system interrupts, effectively halting thread preemption in the system[14][18], while critical tasks execute. However, to make this work, you cannot implement more than two thread priorities, and critical sections where resources are locked need to be very brief and tightly controlled.

Another solution is to implement priority ceilings, or priority boosting[16], where a lower priority thread that acquires a lock has its priority temporarily increased to help ensure that it will complete its execution, and release its lock, as quickly as possible. However, a more practical and less-invasive solution is to implement the priority inheritance protocol.

With priority inheritance, the system code that implements resource locking checks to see if a lower priority thread already owns a lock on the associated resource when a thread attempts to lock it. If one does, that owning thread's priority is temporarily increased to match that of the higher priority thread attempting to acquire the lock. As a result, the lock owner (once blocked at lower priority) will execute, release the lock, and then be restored to its original priority level.

Going back to our original example, priority inheritance would effectively boost thread $\top_2$'s priority to equal that of thread $\top_1$'s, where thread $\top_3$'s would continue to block,

ISSN (Online): 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

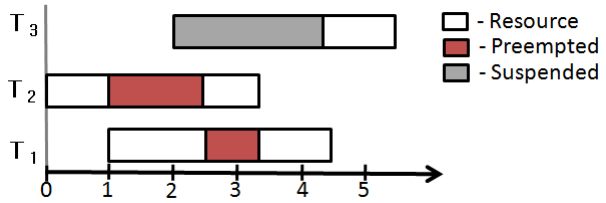

Figure 8: Solution for Priority Inversion problem

allowing $\top_2$ to release its lock sooner. In Fig. 8, you can see how this allows thread $\top_1$ to resume sooner, without the unbounded latency caused by thread $\top_3$'s unknown execution time. Once the lock on R is released by thread $\top_2$, its priority is restored to its original value and the system executes according to normal priority-based rules.

B. Cooperative Scheduling

Generally in real time environment a task may be preempted by hard real time tasks i.e., temporarily suspending a task to switch to a higher priority task and later resuming. In such a case Cooperative Scheduling allows tasks to be scheduled through the use of a background periodic timer that creates a system tick. The difference is that rather than having priorities and preemption, the cooperative scheduler only executes tasks that occur at a time periodic interval. If two tasks are due to run at the same time, the task higher up in the task list runs first followed by the second and so on. During Scheduling the scheduler should use a single interrupt driven timer to keep track of system time[1][7]. The advantage of using Cooperative Scheduling are as follows,

- It increases the schedulability of the system, and it can lead to lower values of computation.
- While preemption no interference will occur during the last slot of execution.

Let the execution of last block be F_i, then the Eq. 12 becomes,

$$w_i^{n+1} = C_i + B_{max} - F_i + \sum_{j \in hp(i)} \in hp(i) \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j \quad (21)$$

As we discuss in Algorithm 1, response time $R_i = w_i^n$ when $w_i^n == w_i^{n+1}$. Now if we consider the Cooperative Scheduling the response time[10] becomes,

$$R_i = w_i^n + F_i \quad (22)$$

C. Analysis of Jitter

Jitter[17] is nothing but the average difference between the arrival and actual execution of task. Every time the difference is pretty different so its very difficult to decide how much jitter we have to consider?. That's all scenario we already discuss in the literature survey, here we analyze an example which completely cover the concept of jitter. Consider the same task $\top_1(2,5)$ and $\top_2(4,7)$, and the response time was already calculated in the section IV; details as shown in the below table-II.

TABLE II: Tasks parameters values

$Task$	T	D	C	R
$\top_1$	5	5	2	2
$\top_2$	7	7	4	7

What happens if there is a difference between invocation time and execution time? Let us have a look to below fig. 9 and suppose the following happens,

- A task $\top_1$ is invoke at $t = 0ms$, but as delay occurs of $2ms$ before execution. So the task $\top_1$ runs at time $t = 2$.
- At time $t + 2$ task $\top_2$ get released but it is preempted by task $\top_1$ because of its higher priority.
- In the fig. 9 it is clearly seen that $\top_2$ misses its deadline and get preempted by $\top_1$ with jitter of just $1ms$ and invoked at $t + 8$. So task $\top_1$ finish its execution but task $\top_2$ misses its deadline[18]. task $\top_2$ is preempted by two times because $R_2 > T_1 - J_1$ i.e., $(7 > 5 - 1)$ as discuss in literature survey of subsection H[19].

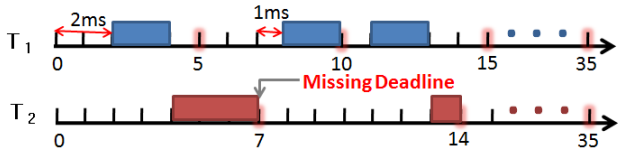

Figure 9: Example to show Jitter during execution

VI. Conclusion & Future Scope

Studying the schedulability test for both dynamic as well static conditions. We analyze the utilization as primary predictability to test the schedulability, but as we proved that some time the test is not optimal. For optimal schedulability we find response time analysis which gives optimal scehdulability test. But during analysis while task interaction and resource sharing find some loop holes like when any higher priority task is get hold back by lower priority task cause blocking that because of priority inversion but using priority ceiling we showed that it can minimize the effect of priority inversion. Also we discuss cooperative scheduling that showed how it can reduce the WCET time in some extend. In the last a major issue called jitter was discuss and proved with example that how it affects the schedulability test even if the tasks are schedulable and because of its unpredictability we used to take difference of two consecutive jitter.

Some more issues are their which affects the schedulability test and need to consider for optimal scheduling like arbitrary deadline .i.e, when $D > T$, non-optimal analysis like including offset to protect task from missing its deadline. That all remaining issues are put for future scope.

References

[1] Swati Pandit, Rajashree Shedge, Survey of Real Time Scheduling Algorithms,e-ISSN: 2278-0661, p- ISSN: 2278-8727 Volume 13, Issue 2 (Jul. - Aug. 2013), PP 44-51

[2] Comparison of Different Task Scheduling Algorithms in RTOS A Survey by Dr. D. G. Harkut and Prof. Anuj M. Agrawal, Volume 4, Issue 7, July 2014 ISSN: 2277 128X.
[3] Preempt a Job or Not in EDF Scheduling of Uniprocessor Systems by Jinkyu Lee, Member, IEEE, and Kang G. Shin, Life Fellow, IEEE TRANSACTIONS ON COMPUTERS, VOL. 63, NO. 5, MAY 2014
[4] Rate Monotonic vs EDF Judgment Day by Giorgio C. Buttazzo, Real-Time Systems, 29, 526, 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands
[5] Instantaneous Utilization Based Scheduling Algorithms for Real Time Systems by Radhakrishna Naik, R.R.Manthalkar, Radhakrishna Naik et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (2) , 2011, 654-662
[6] Fixed-Priority Multiprocessor Scheduling with Liu Laylands Utilization Bound by Nan Guanyz, Martin Stiggey, Wang Yiyz and Ge Yu, sponsored by CoDeR-MP, UPMARC, and NSF of China under Grant No. 60973017 and 60773220.
[7] T. Bijlsma, "Performance of Real-Time Scheduling on Sensor Nodes", July 6, 2006
[8] C.L. LIU AND J. W. LAYLAND, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment" Journal of the Association for Computing Machinery, Vol 20, No. 1, January 1973 pp. 46-61.
[9] On Improving Real-Time Interrupt Latencies of Hybrid Operating Systems with Two-Level Hardware Interrupts by Miao Liu, Duo Liu, Yi Wang, Meng Wang, and Zili Shao, Member IEEE, IEEE Transactions on Ccomputers, vol. 60, no. 7, july 2011
[10] Feasibility Analysis of Real-Time Periodic tasks with offsets by Rodolfo Pellizzoni and Giuseppe Lipari, Real-Time Systems, 30, 105128, 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.
[11] Jian-Jia Chen, Samarjit Chakraborty, Resource Augmentation Bounds for Approximate Demand Bound Functions, 1052-8725/11 2011 IEEE DOI 10.1109/RTSS.2011.32
[12] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System", The Computer Journal (1986) 29 (5): 390-395. doi: 10.1093/comjnl/29.5.390 2015
[13] Ozalp Babaoglu, Keith Marzullo and Fred B. Schneider, "A Formalization of Priority Inversion", Real-Time Systems, 5, 285-303 (1993)
[14] Tarek Helmy & Syed S. Jafri, Avoidance of Priority Inversion in Real Time Systems Based on Resource Restoration,International Journal of Computer Science Applications, Vol. III, No. I, pp. 40 - 50
[15] A. Datum, J. Reisinger, W. Schwabl, and H. Kopetz, The Real-Time Operating System of MARS, Vienna, 1988-10-13
[16] Bruno Dutertre, "The Priority Ceiling Protocol Formalization and Analysis Using PVS", October, 1999
[17] Zvika Brakerski and Boaz Patt-Shamir, "Jitter-Approximation Tradeoff for Periodic Scheduling", Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS04), 0-7695-2132-0/04/2004.
[18] Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor by Sanjoy K. Baruah, pp-182-190, Orlando, Florida December 1990 IEEE. Aloysius K. Mok, and Louis E. Rosier IEEE Computer Society Press.
[19] M. Piaggio, A. Sgorbissa, and R. Zaccaria, Pre-emptive versus non-preemptive real time scheduling in intelligentmobile robotics, J. Exp. Theor. Artif. Intell., vol. 12, no. 2, pp. 235245, 2000.