Enhancement in System Schedulability by Controlling Task Releases

Basharat Mahmood
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan

Saif ur Rehman Malik
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan

Naveed Ahmad
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan

Adeel Anjum
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan

***Abstract*—In real-time systems fixed priority scheduling techniques are considered superior than the dynamic priority counterparts from implementation perspectives; however the dynamic priority assignments dominate the fixed priority mechanism when it comes to system utilization. Considering this gap, a number of results are added to real-time system literature recently that achieve higher utilization at the cost of tuning task parameters. We further investigate this problem by proposing a novel fixed priority scheduling technique that keeps task parameters intact. The proposed technique favors the lower priority tasks by blocking the release of higher priority tasks without hurting their deadlines. The aforementioned strategy helps in creating some extra space that is utilized by a lower priority task to complete its execution. It is proved that the proposed technique dominates pure preemptive scheduling. Furthermore the results obtained are applied to an example task set which is not schedulable with preemption threshold scheduling and quantum based scheduling but it is schedulable with proposed technique. The analyses show the supremacy of our work over existing fixed priority alternatives from utilization perspective.**

***Keywords*—Real-time Systems; Fixed Priority Scheduling; RM Scheduling; Priority Inversion**

I. INTRODUCTION

Real-time systems are built to execute temporally constrained tasks. On such platforms, the accuracy of a system depends not only upon the correctness of response, but also the time these results are obtained. Missing a task deadline may result in serious damage, especially in hard real-time systems [8]. It is not a must for a real-time system to be very fast, but it must be enough capable to execute its tasks within a specified time. Priority assignment to real-time tasks, is the process of deciding the order in which different tasks are executed. In real-time scheduling, the scheduler is responsible to allocate tasks on the processor in such a way that all timing constraints are satisfied.

Fulfilling the timing constraints of tasks is essential to real-time systems and hence the scheduling problem plays an important role in real-time systems theory. The fixed priority scheduling technique is widely used in real-time systems due to its simplicity and predictability. Under fixed priority class, Rate-monotonic (RM) algorithm is a well-known fixed priority assignment algorithm. It is an optimal algorithm for the implicit deadline model [10] [2]. The RM scheduling algorithm is subdivided into two main streams, preemptive scheduling and non-preemptive scheduling. In preemptive scheduling, a lower priority task is preempted when a higher priority task is released, while non-preemptive scheduling does not allow such preemptions. Generally, preemptive scheduling provides better schedulability than non-preemptive scheduling, but it is not always the case. Both preemptive and non-preemptive scheduling fail to guarantee 100 % CPU utilization.

Lot of efforts have been made recently to improve the schedulability of fixed priority scheduling. Different variants of preemptive scheduling have been proposed, which use the concept of priority inversion in order to improve the schedulability. These techniques allow a lower priority task to block a higher priority task. Deferred preemption [19], Preemption threshold scheduling [2] and Quantum based scheduling [5] are examples of such techniques, which improve the schedulability of fixed priority scheduling by priority inversion at runtime.

In this paper a new fixed priority scheduling technique named as CTR is proposed. The CTR technique blocks the task releases for a predefined interval of time without hurting their deadline in order to create some extra space for the currently executing task. In CTR technique, each task is assigned a feasible release block time. At runtime, tasks are kept in block state at their actual release times and are released after their assigned block time. In this way, some extra space could be created for the lower priority tasks to execute. It is proved that such blockage of task releases, does not hurt the deadline of tasks. It is also proved that the CTR technique dominates the RM preemptive scheduling in terms of schedulability. A task set is also given which is not schedulable with preemption threshold scheduling and quantum based scheduling but, it is schedulable with the CTR technique. This shows that the CTR technique has at least an incomparable relation with these techniques.

A. Related Work

In 1973, Liu and Layland did the pioneer and the most influential work in real-time scheduling theory. In their seminal paper [10], they proposed an optimal fixed priority assignment algorithm called rate-monotonic algorithm for the implicit deadline task model. In the same paper, they derived a sufficient schedulability test called LL-bound to predict the feasibility of the system. After that, a lot of work has been done to improve the system feasibility prediction. This work is mainly of two types, the exact schedulability tests [16][15][3][9] and sufficient schedulability tests[10][12].

To reduce the run-time overhead due to task preemptions, limited-preemptions model has been proposed [21][22]. In this model each task is divided into a number of non-preemptive regions and is considered non-preemptive within those regions. These regions may be either fixed or floating [22]. Under fixed pre-emption point model, the non-preemptive regions are predefined while in floating preemption point model the location of non-preemptive regions are un-known.

Different variants of RM preemptive scheduling have been proposed in literature, to improve the schedulability [20][19] [2][5]. In [20] dual priority scheduling model is presented. In this model each task is executed in dual phases with different static priorities. The transition from one phase to another is made at fixed points. This model dominates the RM scheduling but, is considered not viable due to its complexity.

The deferred preemptions scheduling technique [19] assigns each task τ_i an interval q_i for which it remains non-pre-emptible. At runtime when a higher priority task is released, it is kept blocked if the lower priority task is in non-preemptible section otherwise it is preempted.

Preemption threshold scheduling [2] is a dual priority scheduling technique. It assigns each task a regular priority and a preemption threshold value which is greater or equal to its priority. At runtime when a task is executed, its priority is raised to its preemption threshold value. In this way a task can block those higher priority tasks whose priority is less than its preemption threshold value. Preemption threshold scheduling dominates preemptive and non-preemptive scheduling in terms of schedulability.

Another variant of preemptive scheduling is the quantum based scheduling [5]. In quantum based scheduling, CPU time is divided into discrete units called quanta. At runtime, the CPU time is allocated to tasks in the form of quantum. When a quantum is allocated to the task, that task cannot be preempted until the quantum expires or task is completed. Both preemption threshold scheduling and quantum based scheduling improve the schedulability of fixed priority scheduling, but still fail to guarantee 100 % utilization.

In [23] ready-Q locking mechanism is proposed. Ready-Q locking improves the schedulability by locking the ready queue at runtime in order to reduce the interference from higher priority task during the execution of a lower priority task. For further improvement in schedulability, preemption threshold scheduling is also merged with ready-Q locking mechanism [23].

B. Contribution of the Paper:

This work has the following contributions

- A novel fixed priority scheduling technique CTR scheduling is proposed in this paper. The CTR scheduling controls the task releases in order to enhance schedulability
- It is proved that the CTR scheduling dominates RM preemptive scheduling in terms of schedulability
- It is also proved that the CTR scheduling has at-least an incomparable relation with preemption threshold scheduling and quantum-based scheduling
- The improvement in schedulability is also shown by experiments on synthetic task sets

C. Paper Organization:

The rest of the paper is organized as follows. In section II, the system model and basic terminologies are discussed. In section III, the proposed technique is explained in detail with examples. The proposed technique is compared with existing techniques in section IV and finally our work is concluded in section V.

II. System Model, Assumptions and Notations

A. Task Model

We consider the classical periodic real-time task model. Each task τ_i is defined by the tuple (C_i, P_i, D_i). Each task consists of a sequence of infinite jobs. The time at which the first job of a task is released is called its phase. If the phase of a task τ_i is Φ_i then the k^{th} job of τ_i is released at $J_{i,k} = \varphi_i+(k-1)*P_i$. The absolute deadline of the k^{th} job of task τ_i is $d_{i,k} = r_{i,k} + D_i$. The portion of CPU time used by a task τ_i is called its utilization and can be calculated by C_i/P_i. The total utilization of the system can be determined by $U_p = \sum_{i=1}^{n} U_i$.

B. System Model:

We consider a real-time system which consists of a single processor. The workload for the system is defined by set τ of n tasks. A fixed priority scheduler is used to schedule tasks. The scheduler assigns priorities to tasks according to RM algorithm. Each task τ_i is assigned a feasible release block time (Δr_i). When the k^{th} job of τ_i is released at $r_{i,k}$, it is considered in blocked state for the interval ($r_{i,k}$, $r_{i,k} + \Delta r_i$) and is released at $r_{i,k} + \Delta r_i$ time. During block state a task can only be executed if the CPU is free.

C. Assumptions:

We consider the following assumption for our technique

(A1) Task sets follow the implicit deadline model. It means that for any task (τ_i), relative deadline is equal to its period i.e. $D_i = P_i$ therefore, a task can be simply defined by (C_i, D_i)

(A2) Workload is defined by a set τ of n tasks and all tasks in τ are periodic

(A3) It is assumed that only computational resources are required to execute a task and all other resources are negligible

(A4) Any task can be preempted at any time and no task has any non-pre-emptible part

(A5) All tasks are independent and no precedence constraints exist among them

(A6) All runtime costs are negligible

The notations used are shown in TABLE I

TABLE I. NOTATIONS USED AND MEANINGS

Notation	*Meaning*
τ	Set of tasks
τ_i	Task i, $\tau_i \in \tau$
C_i	Worst case execution time of τ_i
P_i	Period of τ_i
R_i	Worst case response time of τ_i under RM scheduling
E_i	The time period during which a task τ_i is released and completes its execution under CTR scheduling
U_i	Utilization of τ_i
D_i	Relative deadline of τ_i
$J_{i,k}$	K_{th} job of τ_i
$d_{j,t}$	Absolute deadline of τ_j at time t
t	Current time
Φ_i	Release time of first job of τ_i
r_i	Feasible release block time of τ_i
$r_{i,k}$	Release time of k_{th} job of τ_i
$t_{i,t}^{avl}$	Time available to τ_i at time t
$t_{i,t}^{req}$	Time required to τ_i at time t
τ_t^{exe}	Task executing at time t
τ_t^{rel}	Task released at time t
$\tau_{r,t}^{h}$	Highest priority ready task at time t
$C_{i,t}^{rem}$	Remaining execution time of task τ_i at time t
R[ ]	Queue of released tasks
J[]	Queue of tasks
P (τ_i)	Priority of τ_i
$I_{(t,d_{i,t})}^{hp(i)}$	Interference from tasks having higher priority than τ_i during (t, $d_{i,t}$)
$e_{j,t}$	Extra space created for τ_j at time t due to the delay in release of higher priority tasks

III. CONTROLLED-TASK-RELEASES (CTR) REAL-TIME SCHEDULING

The CTR scheduling technique is discussed in following subsections in detail

A. Overview of the Technique

If we analyze the runtime behavior of RM preemptive scheduling, it is observed that when a higher priority task τ_i is released at time t, currently executing job of lower priority task τ_j is preempted immediately. Now, τ_j misses its deadline if

$$d_{j,t} - t \leq C_{j,t}^{rem} + I_{(t,d_{j,t})}^{hp(j)}$$

where $C_{j,t}^{rem}$ is the remaining execution time of τ_j at current time t, $d_{j,t}$ is the absolute deadline of currently preempted job of τ_j , t is the current time and $I_{(t,d_{j,t})}^{hp(j)}$ is the interference from tasks with higher priority than τ_j during (t, $d_{j,t}$) interval.

Such tasks can be made schedulable if the release of higher priority task is delayed without hurting its deadline. Such delays create some extra space for the lower priority tasks to complete their execution. The CTR technique utilizes this idea to improve the schedulability.

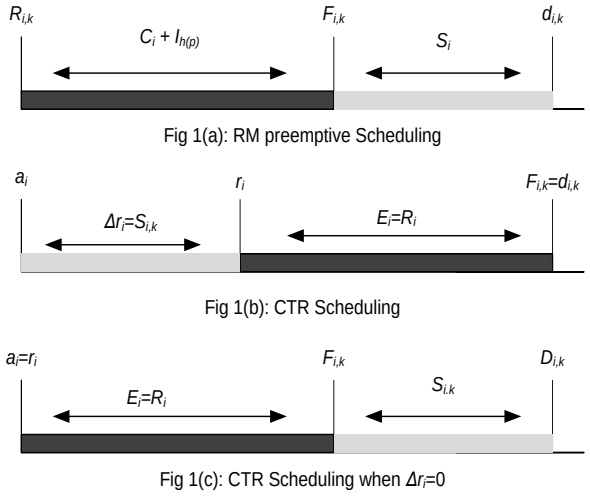

Fig. 1. Comparison of runtime behavior of RM preemptive and CTR scheduling (a) RM scheduling (b)CTR scheduling (c) CTR scheduling with Δri = 0

The CTR scheduling assigns each task τ_i a feasible release block time (Δr_i). At runtime, a task τ_i is considered in blocked state at its actual release time and remains in this state for Δr_i time. After Δr_i time τ_i is released and its priority is compared with the currently executing task. If τ_i has higher priority than the currently executing task, then the task is preempted otherwise it continues. At runtime, if the CPU is free and no task is executing then the lowest priority blocked task is executed.

In Fig 1, the runtime behavior of CTR scheduling is compared with RM preemptive scheduling. Fig 1(a) shows the execution of τ_i with RM scheduling. The k_{th} job of τ_i is released at $r_{i,k}$ and completes its execution at $F_{i,k}$ ahead of its deadline $d_{i,k}$. This difference is shown by $S_{i,k}$ where $S_{i,k} = d_{i,k} - (C_{j,t}^{rem} + I_{(t,d_{j,t})}^{hp(j)})$. On the other hand, the proposed technique blocks the release of task τ_i for Δr_i (or $S_{i,k}$) amount of time therefor, it completes at its deadline as shown in Fig 2(b). The CTR scheduling behaves similarly to RM preemptive scheduling if the release of the task τ_i is not blocked i.e. in $\Delta r_i = 0$ (shown in Fig 1(c)).

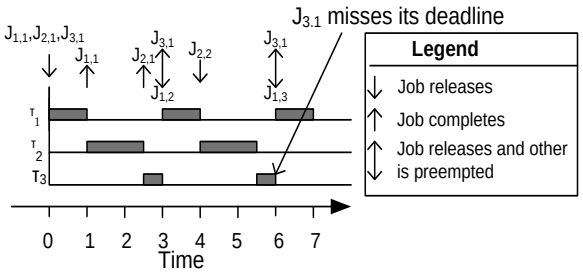

Fig. 2. RM preemptive scheduling of task set given in TABLE II, τ_3 misses its deadline

B. Motivational Example

The following example illustrates the benefits of CTR scheduling. The tasks and their attributes for this example are given in TABLE II.

TABLE II. EXAMPLE TASK SET

Task(τ_i)	WCET(C_i)	Period(P_i)	Release-Block time (Δr_i)
τ_1	1	3	2
τ_2	1.5	4	0.5
τ_3	1.5	6	0

If we apply the RM preemptive scheduling on the example task set given in TABLE II, then the task set is not schedulable because the deadline of τ_3 is missed. The scheduling of task set with RM preemptive scheduling is shown in Fig 2.

Now, if we apply the CTR scheduling on the same task set by assigning each task a feasible task release-block time as given in TABLE II, then the task is schedulable (The method of assigning release-block time is discussed in the following subsection). TABLE III shows the sequence of jobs in which they are activated and released under the task release mechanism of the CTR scheduling. The runtime behavior with the CTR technique is shown in Fig 3.

TABLE III. ACTIVATION AND RELEASE SEQUENCE OF TASKS UNDER CTR SCHEDULING

Task Jobs($J_{i,k}$)	Activation time	Release time
$J_{1,1}$	0	2
$J_{2,1}$	0	0.5
$J_{3,1}$	0	0
$J_{1,2}$	3	5
$J_{2,2}$	4	4.5
$J_{1,3}$	6	8
$J_{3,2}$	6	6
$J_{2,3}$	8	8.5
$J_{1,4}$	9	11

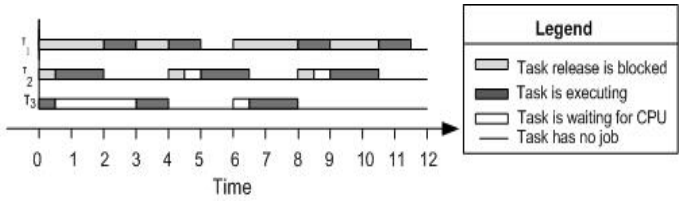

Fig. 3. CTR scheduling of task set given in TABLE II, task set is schedulable

The execution sequence with the proposed technique is explained below

- At t=0, τ_1 and τ_2 are active, but τ_3 is released. Therefore τ_3 is started
- At t=0.5, τ_2 is released. As τ_2 has higher priority than τ_3, therefore τ_3 is preempted and τ_2 starts
- At t=2, not only τ_2 is completed, but also τ_1 is released. So τ_1 starts
- At t=3, τ_1 completes its execution. As at t=3, only τ_3 is ready which was preempted earlier , therefore τ_3 starts and is completed at t=4
- At t=4, no task is ready, but τ_1 and τ_2 are active. As the CPU is free therefore the highest priority active task (τ_1) is assigned to the CPU
- At t=5, τ_1 completes its execution and τ_2 starts

The process continues in a similar way.

C. Assignment of Feasible Release-block Time

The feasible release-block time (Δr_i) for a task τ_i is the difference between the available time to it and the maximum time it may require in worst case. The available time to a task τ_i released at time t is

$$t_{i,t}^{avail} = t - d_{i,t}$$

$$\Longrightarrow t_{i,t}^{avail} = D_i \quad (1)$$

The maximum required time to a task τ_i in the worst case is

$$t_{i,t}^{req} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{C_j} \right\rceil \times C_j \quad (2)$$

Now, the feasible release-block time (Δr_i) value for a task τ_i is

$$\Delta r_i = t_{i,t}^{avail} - t_{i,t}^{req}$$

$$\Rightarrow \Delta r_i = D_i - C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{C_j} \right\rceil \times C_j \quad (3)$$

The Algorithm 1 to assign Δr_i values to tasks is given below

Algorithm 1: assigning feasible release-block time (Δr_i)

```
Require: Set of n tasks (τ)
for i = 1 → n − 1
do
    Δr_i = D_i − C_i + Σ_{j=1}^{i−1} ⌈D_i / C_j⌉ × C_j
    if Δr_i < 0 then
        Δr_i = 0
    End if
end for
```

Theorem1: *Given a task $\tau_i \in \tau$ released at time t, if τ_i is schedulable under the RM preemptive scheduling then it remains schedulable even if its release is blocked for Δr_i time.*

Proof: The given task τ_i released at time t is RM schedulable then

$$t_{i,t}^{avl} \geq t_{i,t}^{req}$$

$$\Rightarrow t_{i,t}^{avl} \geq R_i$$

Where R_i is the worst case response time of τ_i under RM preemptive scheduling and can be calculated by using Response-Time Analysis [3]. When the release of τ_i is blocked for Δr_i time, then the available time to τi is reduced to

$$t_{i,t}^{avl} = D_i - \Delta r_i$$

Now, τ_i remains schedulable if following inequality is satisfied

$$D_i - \Delta r_i \geq R_i$$

We solve the above inequality

$$D_i - (D_i - C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{C_j} \right\rceil \times C_j) \geq R_i$$

$$\Longrightarrow D_i - D_i + C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{C_j} \right\rceil \times C_j \geq R_i$$

$$\Rightarrow C_i + \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{C_j} \right\rceil \times C_j \geq R_i$$

The above condition always remains true. Hence, it is proved that blocking the release of a RM task schedulable for Δr_i time does not hurt its schedulability.

D. Scheduling Algorithm

Now, we present the CTR scheduling algorithm. Initially, priorities are assigned to tasks according to RM algorithm. Then each task is assigned a feasible release-block time. At the start, all tasks with positive Δr value remain in blocked state and the highest priority ready task with zero Δr is assigned to the processor. When a task τ_i is released after remaining in blocked state for Δr_i time, its priority is compared with the priority of the currently executing task τ_j and if τ_i has higher priority than τ_j, then τ_j is preempted otherwise it continues. During runtime if a task is completed and there is no ready task, then the first task in blocked state is assigned to the processor. The scheduling process under the CTR scheduling is shown by algorithm 2.

E. Implementation of the CTR Scheduling

In this section, we discuss the implementation of CTR scheduling and the performance overheads associated with it.

1) Implementation

The implementation of mechanism to block the release of tasks at runtime is the core issue in the implementation of CTR scheduling. For this, the scheduler is required to distinguish the active and ready states of a task at runtime. In order to have better synchronization of task parameters, the job table is required to keep information about activation time and release time of jobs. The CTR scheduler requires two task queues named as job queue and ready queue. The job queue keeps the jobs which are next to release. These jobs are ordered by earliest to release first basis. The ready queue keeps the jobs which are ready to execute, but waiting for CPU due to the execution of a high priority job. These jobs are kept by descending order of their priorities. At run time, the scheduler sets the timing hardware to interrupt the CPU at the release time of the job at the head of the job queue. When an interrupt is generated the scheduler moves all the jobs with the same release time as of the interrupt time, to the ready queue and updates the timing hardware. Under CTR scheduling, when the ready queue is empty the scheduler is required to move an active job to the ready queue. To do this, the scheduler finds the first job in the job queue whose activation time is before or the same as of the current time and moves it to the ready queue.

2) Overheads

Here we discuss the overheads associated with CTR scheduling and compare with other scheduling techniques. The first major overhead associated with CTR scheduling is the assignment of release block time to tasks. This assignment is done off-line and has an O(n) complexity. As this assignment is made off-line therefore it does not cause any performance cut at runtime. No such assignment is required in RM preemptive scheduling. On the other hand, preemption

threshold scheduling has a more complex mechanism to assign preemption threshold values which has an $O(n^2)$ complexity. Similarly, in quantum-based scheduling the assignment of feasible quantum size also has and $O(n^2)$ complexity.

Algorithm 2: *The CTR scheduling*

```
Require: Set of n tasks (τ)
R[ ] = φ   // Queue that holds the released tasks
J[ ] = φ   // Queue that holds the active or blocked tasks
i, k=0

RM(τ )        // Assign priorities to tasks according to RM algorithm

Assign-Delays(τ)     //Assign feasible task release-block times
At t=o:
 CPU ⟷ τ^h_{r,t}  //Highest priority ready task gets the CPU
Upon task activation:
    if (Δr_cur > 0)
       J[ i + 1] ⟷ τ_t^cur
    End if
Upon task completion:
if (R[ ] ≠ φ)   // if ready queue is not empty
  CPU ⟷ τ^h_{r,t}
End if
else if (J[ ] ≠ φ)  //if the queue holding the blocked tasks is not
empty
  CPU ⟷ J[1]   // Assign first task from the blocked task to the CPU
End else if
    else
         wait for task release
    End else

Upon task release:
    if (P(τ_t^exe)<P(τ_t^rel)  //compare the priority of currently
released task with the executing task
         CPU ⟷ τ_t^rel
    End if
    else
         CPU ⟷ τ_t^exe
    End else
```

The second major overhead, which incurs at runtime is that, in CTR scheduling when there is no task in ready queue the scheduler is required to search the job queue to find an active task to execute. It does not affect the performance much because in heavily loaded systems it is very rare to have an empty ready queue.

IV. Evaluation Of The CTR Scheduling

In this section we compare the CTR scheduling with other techniques. We have proved the dominance of CTR scheduling over RM preemptive scheduling in schedulability perspective. This dominance is also validated by experiments on synthetic task sets. The incomparable relation of CTR scheduling with Preemption threshold scheduling and Quantum-based scheduling has also been proved and validated by experiments.

A. Dominance of CTR scheduling over RM preemptive scheduling

RM preemptive scheduling favors high priority task because it immediately preempts the lower priority task when a higher priority task is released. Such early preemptions can cause deadline to miss for lower priority tasks. It is tried in the CTR scheduling to overcome this deficiency by delaying preemptions feasibly. The CTR scheduling creates extra space for lower priority tasks by delaying the release of higher priority task. As a result, it provides better schedulability than RM preemptive scheduling. In following theorem, we prove that the CTR scheduling dominates RM preemptive scheduling in schedulability perspective. It means that CTR scheduling can feasibly schedule all those task sets which are schedulable with RM preemptive scheduling, but it is not guaranteed that RM preemptive scheduling can schedule all the task sets which are schedulable with CTR scheduling.

Theorem 2: *Given a task set τ consisting of n independent, periodic tasks whose deadlines are equal to their periods. If τ is RM schedulable then it is always schedulable with the CTR scheduling technique while the vice versa is not true always.*

Proof: Suppose a lower priority task τ_j is executing at time t and a higher priority task τ_i is released. We discuss the schedulability of both tasks under RM scheduling and CTR scheduling

Case 1: (Schedulability of τ_i) If τ_i is schedulable with RM preemptive scheduling then it is also schedulable with CTR scheduling (by Theorem 1).

Case 2: (Schedulability of τ_j) If τ_j is schedulable with RM preemptive scheduling then it gets its required time before the deadline of its current job. It can be written as

$$d_{j,t} - \mathrm{t} \geq C_{j,t}^{rem} + I_{t,d_{j,t}}^{h(j)}$$

As under the CTR scheduling the release of τ_i is blocked for therefore Δr_i time, it creates some extra space $e_{j,t}$ for τ_j, therefore the above in-equality can be written as

$$d_{j,t} - \mathrm{t} + e_{j,t} \geq C_{j,t}^{rem} + I_{t,d_{j,t}}^{h(j)}$$

Where $e_{j,t} \geq 0$. The above in-equality always remains true because

$$d_{j,t} - \mathrm{t} + e_{j,t} \geq d_{j,t} - \mathrm{t}$$

It shows that if τ_j is schedulable with RM preemptive scheduling then it is also schedulable with CTR scheduling. Now, if τ_j is not schedulable with RM preemptive scheduling then

$$d_{j,t} - \mathrm{t} < C_{j,t}^{rem} + I_{t,d_{j,t}}^{h(j)}$$

Now in similar situation the CTR scheduling creates some extra space $e_{j,t}$ for τ_j by blocking higher priority tasks. Now if

$$d_{j,t} - \mathrm{t} + e_{j,t} \geq C_{j,t}^{rem} + I_{t,d_{j,t}}^{h(j)}$$

then the task is schedulable with the CTR technique and if it holds true for all such situations, then the whole task set is schedulable with the CTR technique. It shows that the CTR technique can schedule some tasks which are not schedulable with the RM technique.

*1) **Experimental Evaluation:***

The CTR Scheduling has been compared with RM preemptive and non-preemptive scheduling, to evaluate the schedulability improvement. For this purpose, we have generated 10^4 task sets. Each task set consists of periodic tasks with $D_i = P_i$. The size of task sets is $n \in \{2, 3, 4, 5, 6, 7, 8, 9\}$ and their period ranges {2, 500}. The utilization of the system was kept from 88% to 100%. Priorities are assigned to tasks by RM algorithm. The performance of different techniques has been evaluated by the percentage of feasible tasks. The results are discussed below.

Fig4 summarizes the experimental results for the CTR, RM preemptive (RMP) and RM non-preemptive (RMNP) scheduling techniques in schedulability perspective. X-axis represents the system's utilization while the Y-axis shows the percentage of feasible task sets. Fig 4(a) shows the results of task sets with n=2 or 3. It can be seen clearly that the CTR scheduling surpasses both RMP and RMNP in schedulability perspective. At lower system utilization levels (88% to 90%) the performance gap is less (less than 10%) but it goes on increasing as we move towards higher system utilization levels. At 100% system utilization, RMP schedule 67% task sets feasibly while the RMNP schedules 58% and CTR scheduling schedules 92% task sets. This decrease in performance of RMP and RMNP occurs due to their extreme behavior towards preemptions which result in deadline miss for low priority tasks. On the other hand CTR scheduling performs better by adopting a more sensible behavior towards preemptions.

In Fig 4(b), the schedulability results are shown for task sets with n=4 or 5. The dominance of CTR scheduling over RMP and RMPNP is clearly observable. At lower system utilization levels, RMP and RMNP perform reasonably well but as the system utilization increases their performance decreases hugely. On the other hand, CTR scheduling out classes both RMP and RMNP techniques. It schedules 10% more tasks than RMP and RMNP at 88% utilization while this gap increases to 30% at 100% system utilization. Fig 4(c) and Fig 4(d) summarize the results for the task set with n= 6 or 7 and n=8 or 9. The dominating performance of CTR scheduling as compared to RMP and RMNP scheduling is again observable.

B. Incomparable relation of CTR scheduling with Preemption threshold scheduling and Quantum-based scheduling

In this section we have compared the CTR scheduling with Preemption threshold scheduling and Quantum-based scheduling. We have shown that CTR scheduling has at-least an incomparable relation with these techniques. It means, there exists at-least one task set which is not schedulable with Preemption threshold scheduling and Quantum-based scheduling but CTR scheduling feasibly schedules it. Consider a task set given in TABLE IV. The given taskset is not schedulable with Preemption threshold scheduling. If the preemption threshold value of τ_3 is 1, then its WCRT is 8 and its deadline is missed. Similarly, at higher preemption threshold values of 2 and 3, τ_3 remains un-schedulable. On the other hand, if we apply CTR scheduling on the same task sets by assigning $\Delta r_1 = 2$, $\Delta r_2 = 0$ and $\Delta r_3 = 0$, the task set becomes schedulable as shown in TABLE IV.

TABLE IV. EXAMPLE TASK SET AND COMPARISON OF WCRT

Task (τ_i)	**WCET(C_i)**	**Period(P_i)**	**WCRT(Preemption threshold)**	**WCRT(CTR Scheduling)**
τ_1	1	3	1	3
τ_2	2	4	2	3
τ_2	1	6	8	4

Quantum-based scheduling also fails to schedule the task set given in TABLE IV. For the give task set, the upper bound and lower bound on quantum size is 2. TABLE V shows that the WCRT of τ_3 is 8 and its deadline is missed, when quantum size is 2.

TABLE V. WCRT OF TASK SET GIVEN IN TABLE IV UNDER QUANTUM-BASED SCHEDULING

Quantum Size	**Task(τ_i)**	**WCRT**
2	τ_1	1
	τ_2	3
	τ_3	8

Therefore, by this example, the at-least incomparable relation between the CTR scheduling and Quantum-based scheduling and Preemption threshold scheduling is proved.

*1) **Experimental Evaluation:***

To evaluate the performance of Preemption threshold scheduling (PTS) and Quantum-based scheduling (QBS) against CTR scheduling, we repeated the experiments given in previous section for these techniques. When PTS and QBS techniques are applied on same task sets, the obtained results are shown in Figure 5. It can be seen that both PTS and QBS performed better than RM preemptive and non-preemptive scheduling but at higher system utilization (95% and above) CTR scheduling dominates.

Fig 5(a) shows the experimental results of task sets with n = {2, 3}. It can be seen that at lower system utilization levels

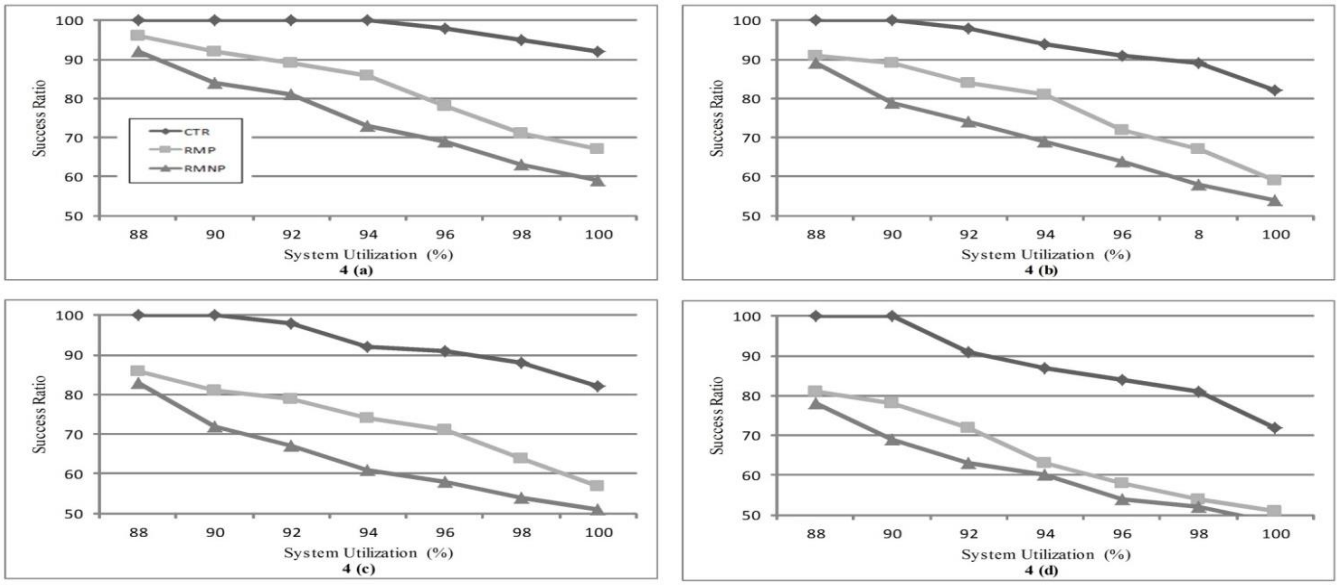

Fig. 4. Performance analysis of RM preemptive (RMP), RM non-preemptive (RMNP) and CTR scheduling on synthetic data sets in schedulability perspective (a) n={2,3} (b) n={4,5} (c) n={6,7} (d) n={8,9}

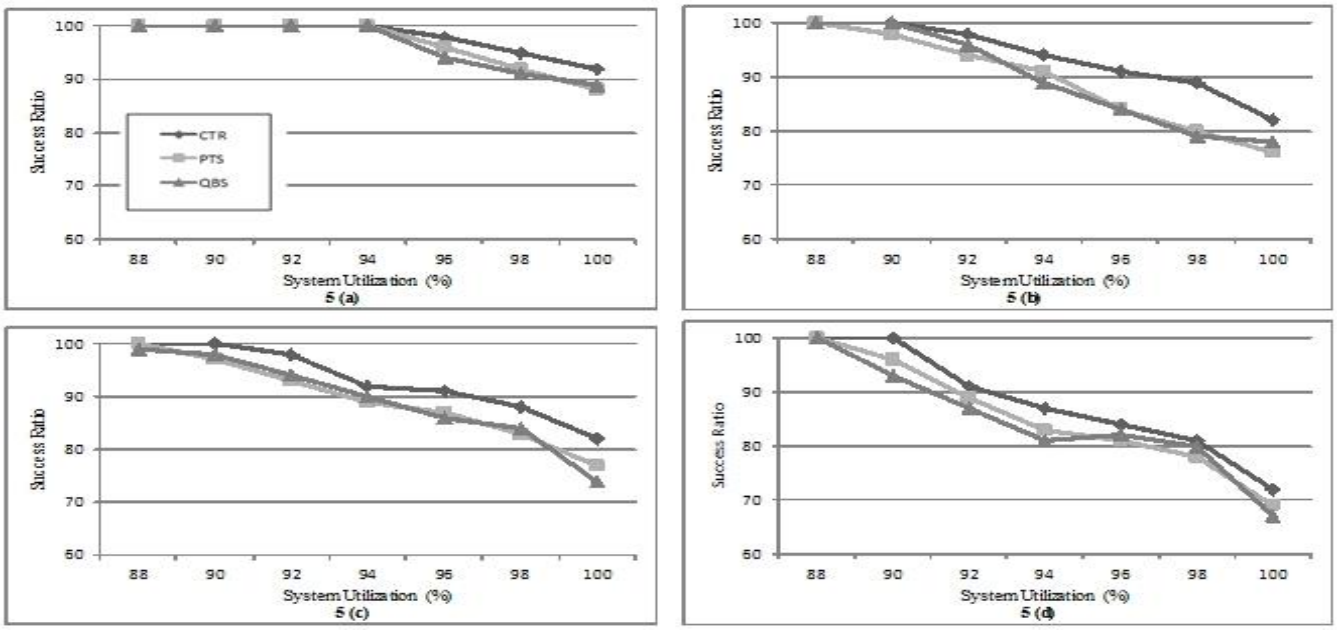

Fig. 5. Performance analysis of CTR, PTS and QBS scheduling on synthetic data sets in schedulability perspective (a) n={2,3} (b) n={4,5} (c) n={6,7} (d) n={8,9}

(88% to 94%) PTS, QBS and CTR scheduling performed very well but as we move further towards higher system utilization the CTR scheduling performs slightly better. At 100% system utilization the CTR scheduling schedules 92% tasks sets feasibly while PTS schedules 88% and QBS schedules 87% task sets. For task sets having 4 or 5 tasks, the obtained results are shown in Fig 5 (b). The similar performance of CTR, PTS and QBS scheduling at lower system utilization levels is easy to observe. At higher system utilization levels, again CTR scheduling performs better than PTS and QBS scheduling. Similar result are also obtained for task sets with n=6, 7 and n=8, 9 and are summarized in Fig 5(c) and Fig 5 (d).

As compared to RM preemptive and non-preemptive scheduling, preemption threshold scheduling and quantum-based scheduling adopt more flexible and wise behavior in preemptions perspective. As a result, preemption threshold scheduling and quantum based scheduling performed better

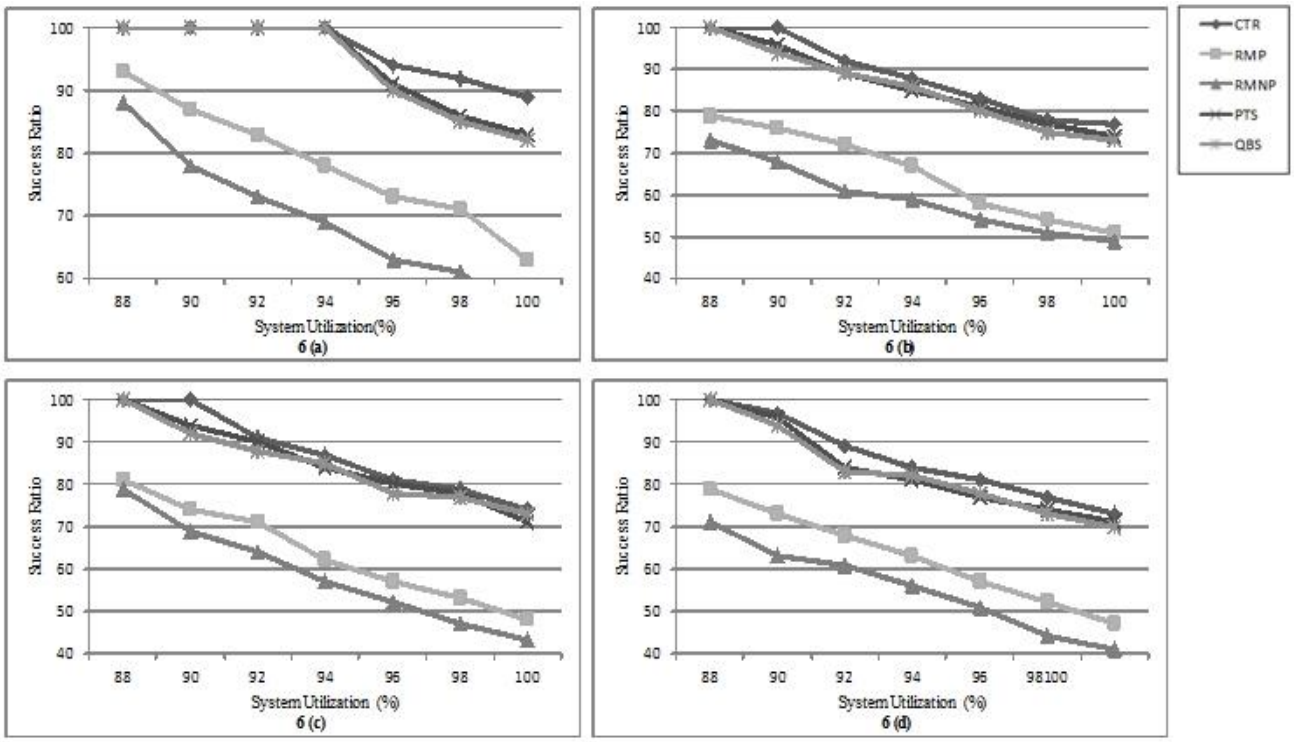

Fig. 6. Performance summary on synthetic constrained and arbitrary deadline task sets in schedulability perspective

but still could not attain the optimal system utilization. On the other hand, CTR scheduling achieves highest system utilization among all because it avoids preemptions till last moment and hence allows low priority tasks to complete, which results in higher utilization. Up-to 95% system utilization Preemption threshold, Quantum based and CTR scheduling showed similar performance but, at higher system utilization [96%, 100%] CTR scheduling achieves highest system utilization.

C. Scheduling constrained and arbitrary deadline tasks

The analysis given in previous sub-sections demonstrates the primacy of CTR scheduling in terms of schedulability over its alternatives but, these results are obtained only for implicit deadline tasks. In this section, we extend the analysis to handle tasks with constrained and arbitrary deadlines. Under the implicit deadline task model, for any task τ_i, D_i is always equal to P_i. This assumption makes the schedulability analysis very simple. However, many practical circumstances require relaxing this assumption. The constrained deadline task model permits D_i to be less than or equal to P_i while in arbitrary deadline model, D_i may be equal to or greater than P_i.

The mechanism to control the release of a task under CTR scheduling does not distress the RM schedulability of a task (by Theorem 1). This result is not specific to the implicit deadline model and remains true for constrained deadline tasks and arbitrary deadline tasks. Furthermore, the dominance of CTR scheduling over RM preemptive scheduling also holds for constrained deadline tasks and arbitrary deadline tasks. Because when a task set is schedulable with RM preemptive scheduling, it is also schedulable with CTR scheduling while a task set which is not schedulable with RM preemptive scheduling may be schedulable with CTR scheduling due to the gain achieved by delaying the task releases.

Corollary 1: *A RM schedulable task set τ, consisting of n periodic, independent, constrained deadline tasks is always schedulable with CTR scheduling but the vice-versa is not always true.*

Corollary 2: *A RM schedulable task set τ , consisting of n periodic, independent, arbitrary deadline tasks is always schedulable with CTR scheduling but the vice-versa is not always true.*

*1) **Experimental Evaluation:***

To evaluate the performance of CTR scheduling for the constrained task model and arbitrary deadline task model, we have created 10^4 task sets for each category. These tasks are generated in a similar way as explained in section IV. The Figure 6 summarizes the results. It can be seen clearly that, RM non-preemptive scheduling performs fine at low system utilization but the instant system utilization exceeds 91% the percentage of feasible task sets under RM non-preemptive scheduling starts decreasing and it tapers down to less than 53% at 100% system utilization. The performance of RM preemptive scheduling is better as compared to RM non-preemptive scheduling due to permitting preemption, but still at high system utilization the percentage of feasible tasks are low. At 100% system utilization, RM preemptive scheduling succeeds to schedule 62% task set feasibly. The performance of preemption threshold scheduling and quantum-based scheduling is comparatively better than RM scheduling, but below than the CTR scheduling at higher system utilization

level.

V. Conclusion and Future Work

Novel results are established for fixed priority scheduling by controlled task releases. The controlled release timings are exploited to improve the schedulability of fixed priority scheduling. It is proved that the pro-posed technique dominates RM preemptive technique in the sense that it schedules all task set that are schedulable with RM preemptive but vice versa is not true. As an example, it is shown that the proposed technique successfully schedule a given task system where preemption threshold scheduling or quantum based scheduling techniques fail. In this paper tasks are restricted to be only periodic; however, as a future work more interesting are expected when applied to sporadic tasks systems.

References

[1] Davis, R.I., Burns, A., Baruah, S., Rothvoss, T., George, L., and Gettings, O., Exact Comparison of Fixed Priority and EDF Scheduling based on Speedup Factors for both Pre-emptive and Non-pre-emptive Paradigms, Real-Time Systems Journal, 51(5), pp566-601, 2015.

[2] Wang, Y., and Saksena, M.: 'Scheduling fixed priority tasks with preemption threshold'. In proceedings of the 6th international conference on real time computing systems and applications, Hong Kong, China, Dec 1999, pp. 328-335

[3] Audsley, N.C., Burns, A., Tindell, K., and Wellings, A.: 'Applying new scheduling theory to static priority preemptive scheduling', Software Engineering Journal, 1993, 8, (2), pp. 80-89

[4] Bini, E., and Buttazo, G.C.: 'The space for Rate Monotonic schedulability'. 23rd IEEE Real-Time Systems symposium, Austin, TX, USA, Feb 2002, pp. 169-178

[5] Park, M.,Yoo, H.J., and Chae, J.: 'Analysis on Quantum-Based fixed priority scheduling of Real-Time tasks'. Proceedings of the 3rd international conference on ubiquitous information management and communication, Suwon, SKKU, Korea, Jan 2009, pp. 627-634

[6] Davis, R.I, A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems, ACM SIGBED review, 11(1), pp. 8-19, 2014.

[7] George, L., Riverre, N., Spuri, M.: 'Preemptive and Non-preemptive Real-Time Uniprocessor Scheduling', Research Report RR-2966, INRIA, France, 1996

[8] Huhang, W.H., Chen, J., Zhou, H., and Liu, C., PASS: Priority Assignment of Real-Time Tasks with Dynamic Suspending Behavior under Fixed-Priority Scheduling, Technical Reports in Computer Science, Dortmund University of Technology, 2015.

[9] Bini, E., and Buttazzo, G.C.: 'Schedulability Analysis of Periodic Fixed Priority Systems', IEEE Transactions on Computers, 2004, 53, (11), pp. 1462-1473

[10] Liu, C.L., and Layland, J.W.: 'Scheduling algorithms for multiprogramming in a hard real-time environment', Journal of the ACM, 1973, 20,(1), pp. 40-61

[11] Tindell, K.W., Bums, A., Wellings, A.: 'An extendible approach for analyzing fixed priority hard real-time tasks', Real-Time Systems Journal, 1994, 6, pp. 133-151

[12] Bini, E., and Buttazzo, G.C.: 'A Hyperbolic Bound for the Rate Monotonic Algorithm'. In Proceedings of the 13th Euromicro Conference on Real-Time Systems, Delft, Netherlands, June 2001, pp. 59-66

[13] Min-Allah, N., Ali, I. , Jian-Sheng, X., Yong-Ji, W.: 'Online Feasibility Analysis with Composite-Deadline'. In Proceedings of the 4th International Conference on Innovations in Information Technology, Dubai, UAE, Nov 2007, pp. 357-361

[14] S. Baruah, V. Bonifaci, G. D'angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. Journal of the ACM, 62(2):14:1–14:33, 2015.

[15] Lehoczky, J.P., Sha, L., Ding, Y.: 'The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior'. In Proceedings of the IEEE Real-Time System Symposium, California, USA, Dec 1989, pp. 166-171

[16] Kim, J.E., Abdelzaher, T., and Sha, L., Budgeted Generalized Rate Monotonic Analysis for the Partitioned, yet Globally Scheduled Uniprocessor Model, Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE, pp. 221-231, 2015.

[17] Sha, L., Abdelzaher, T., Erzen, K., Cervin, A., Baker, T., Burns, A., Buttazzo, G.C., Caccamo, M.,Lehoczky, J., Mok, A.K.: 'Real-Time Scheduling Theory: A Historical Perspective', Real-Time Systems, 28 (2), pp. 101–155, 2004.

[18] Wan-Chen, L., Kwei-Jay, L., Hsin-Wen, W., Wei-Kuan, S.: 'Rate monotonic schedulability tests using period-dependent conditions', Real-Time Systems journal, 37 (2), pp. 123-138, 2007.

[19] Thekkilakattil, A., Dobrin, R., and Punnekkat, S., The limited-preemptive feasibility of real-time tasks on uniprocessors. *Real-Time Syst*, 51(3), pp. 247-273, 2015.

[20] Burns, A., and Wellings, A.: 'Dual Priority Assignment: A Practical Method for Increasing Processor Utilization'. In Proceedings of 5th Euromicro Workshop on Real-Time Systems, Oulu, Finland, June 1993, pp. 48-55

[21] Baruah, S.: 'The limited-preemption uniprocessor scheduling of sporadic systems'. In ECRTS 05, Pro-ceedings of Euromicro Conference on Real-Time Systems, Balearic Islands, Spain, July 2005, pp. 137-144

[22] Buttazzo, G., Bertonga, M., and Yao, G., Limited Preemptive Scheduling for Real-Time Systems. A Survey, IEEE transactions on industrial informatics, 9(1), pp. 3-15, 2013.

[23] Marinho.J, Petters, S.M, and Bertogna, M.: 'Extending Fixed Task-Priority Schedulability by Interference Limitation'. In Proceedings of the 20th International Conference on Real-Time and Network Systems, pp. 191-200, 2012.