Real-Time Scheduling

Priority-Driven Scheduling

Fixed-Priority

Fixed-Priority Algorithms

We consider a set of tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$

Any fixed-priority algorithm schedules tasks of $\mathcal{T}$ according to fixed (distinct) priorities assigned to tasks

We write $T_i \sqsupset T_j$ whenever T_i has a higher priority than T_j

We denote by $T_i\uparrow$ the set of tasks that have the same or higher priority than T_i.

Recall that Fixed-Priority Algorithms might not be optimal

Consider $\mathcal{T} = \{T_1, T_2\}$ where $T_1 = (2, 1)$ and $T_2 = (5, 2.5)$

$U^{\mathcal{T}} = 1$ and thus $\mathcal{T}$ is schedulable by EDF

If $T_1 \sqsupset T_2$, then $J_{2,1}$ misses its deadline
If $T_2 \sqsupset T_1$, then $J_{1,1}$ misses its deadline

Critical Instant – Informally

To be able to further analyze fixed-priority algorithms we need to consider a notion of *critical instant*

Intuitively, a critical instant is the time instant in which the system is most loaded, and has its worst response time

Schedulability of a set of tasks is determined by response times of jobs released at critical instants

Critical Instant – Formally

Definition 1

A **critical instant** t_{crit} of a task T_i is a time instant in which a job $J_{i,k}$ in T_i is released so that $J_{i,k}$ either does not meet its deadline, or has the maximum response time of all jobs in T_i

Denote by W_i the response time of such $J_{i,k}$

Theorem 2

In a fixed-priority system where every job completes before the next job in the same task is released, a critical instant of a task T_i occurs when one of its jobs $J_{i,k}$ is released at the same time with a job from every higher-priority task.

Note that the situation described in the theorem does not have to occur if tasks are not in phase. So we use critical instants either to study tasks in phase, or to get upper bounds on schedulability as follows:

- Set phases of all tasks to zero, which gives a new set of tasks $\mathcal{T}' = \{T'_1, \ldots, T'_n\}$
- Determine the response time w' of the first job $J_{i,1}$ in T'_i

Then $w' \geq W_i$, the response time of a job in $\mathcal{T}$ released at the critical instant

RMa and DM Algorithms (reminder)

- **RM** = assigns priorities to tasks based on their periods
  the priority is inversely proportional to the period p_i
- **DM** = assigns priorities to tasks based on their relative deadlines
  the priority is inversely proportional to the relative deadline D_i

(In all cases, ties are broken arbitrarily.)

We consider the following questions:

- Are the algorithms optimal?
- How to efficiently (or even online) test for schedulability?

Optimality of RM for Simply Periodic Tasks

Definition 3

A set $\{T_1, \ldots, T_n\}$ is **simply periodic** if for every pair T_i, T_k satisfying $p_i < p_k$ we have that p_k is an integer multiple of p_i

Example 4

The helicopter control system from the first lecture

Theorem 5

A set $\mathcal{T}$ of n simply periodic, independent, preemptable tasks with $D_i = p_i$ is schedulable on one processor according to RM **iff** $U^{\mathcal{T}} = \sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$.

i.e. on simply periodic tasks RM is as good as EDF

Optimality of DM (RM) among Fixed-Priority Algs.

Theorem 6

A set of independent, preemptable periodic tasks with $D_i \leq p_i$ that are in phase (i.e., $\varphi_i = 0$ for all $i = 1, \ldots, n$) can be feasibly scheduled on one processor according to DM if it can be feasibly scheduled by some fixed-priority algorithm.

Proof.

Assume a fixed-priority feasible schedule with $T_1 \sqsupset \cdots \sqsupset T_n$.

Consider the least i such that the relative deadline D_i of T_i is larger than the relative deadline D_{i+1} of T_{i+1}.

Swap the priorities of T_i and T_{i+1}.

The resulting schedule is still feasible.

DM is obtained by using finitely many swaps. □

Note: If the assumptions of the above theorem hold and all relative deadlines are equal to periods, then RM is optimal among all fixed-priority algorithms.

Fixed-Priority Algorithms: Schedulability

We consider two schedulability tests:

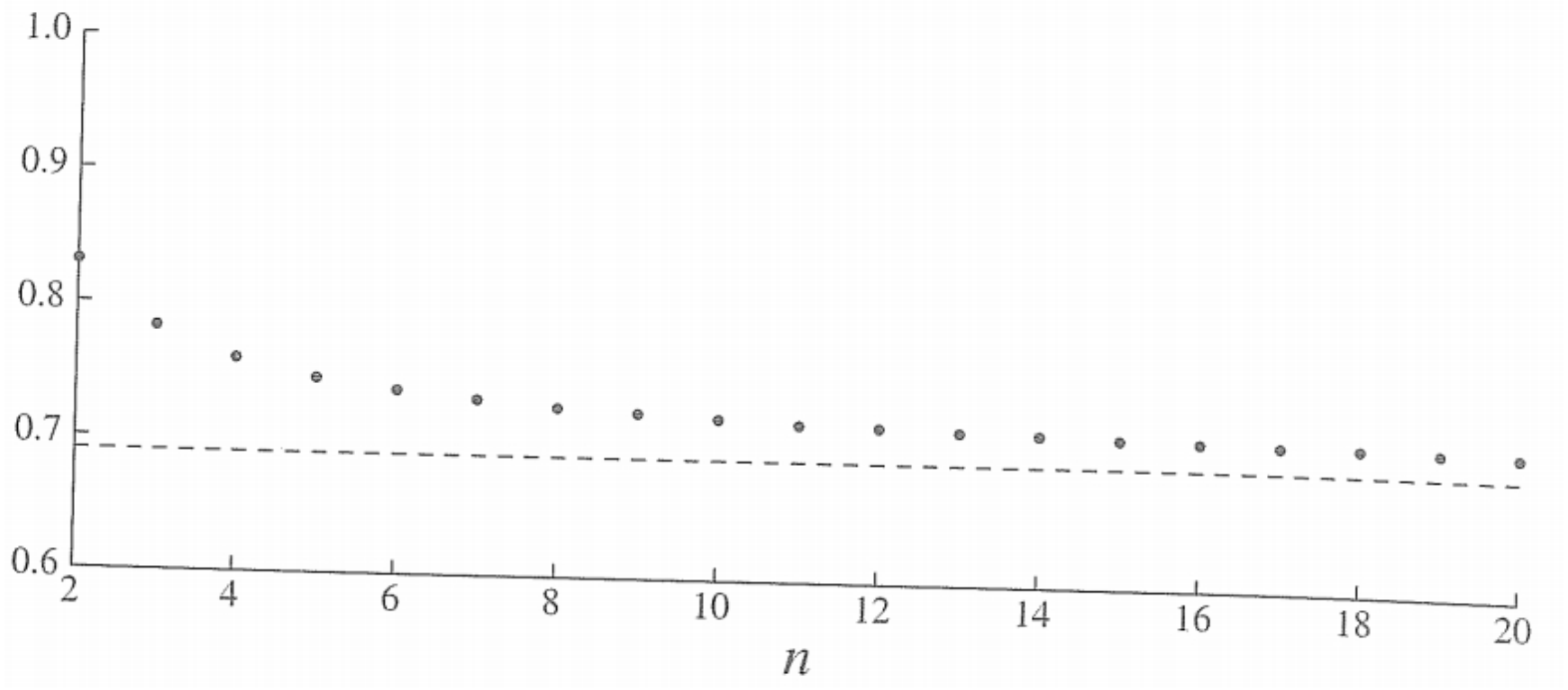
- Schedulable utilization U_{RM} of the RM algorithm.
- Time-demand analysis based on response times of jobs released at critical instants

Schedulable Utilization for RM

Theorem 7

Let us fix $n \in \mathbb{N}$ and consider only independent, preemptable periodic tasks with $D_i = p_i$.

- *If $\mathcal{T}$ is a set of n tasks satisfying $U^{\mathcal{T}} \leq n(2^{1/n} - 1)$, then $U^{\mathcal{T}}$ is schedulable by the RM algorithm.*
- *For every $U > n(2^{1/n} - 1)$ there is a set $\mathcal{T}$ of n tasks satisfying $U^{\mathcal{T}} \leq U$ that is not schedulable by RM.*



Schedulable Utilization for RM

It follows that the maximum schedulable utilization U_{RM} over independent, preemptable periodic tasks satisfies

$$U_{RM} = \inf_n n(2^{1/n} - 1) = \lim_{n \to \infty} n(2^{1/n} - 1) = \ln 2 \approx 0.693$$

Note that $U^{\mathcal{T}} \leq n(2^{1/n} - 1)$ is a sufficient but not necessary condition for schedulability of $\mathcal{T}$ using the RM algorithm (an example will be given later)

In what follows we assume that $p_1 < p_2 < \ldots < p_n$ which implies that $T_1 \sqsupset T_2 \sqsupset \cdots \sqsupset T_n$

Proof Sketch of Theorem 7

A set of tasks $\mathcal{T}$ *fully utilizes* the processor if it is schedulable by RM but any increase in execution time makes the set unschedulable.

Given $n \in \mathbb{N}$, we denote by b_n the greatest lower bound on utilization over all sets of n tasks that fully utilize the processor.

We prove that

$$b_n = n(2^{\frac{1}{n}} - 1)$$

Proof:
Given $p_1, \ldots, p_n$, denote by $U[p_1, \ldots, p_n]$ the minimum utilization over sets of n tasks with periods $p_1, \ldots, p_n$ that fully utilize the processor.

(A) Show that for every set that fully utilizes the processor there is another one whose utilization cannot be larger and

1. is in phase,
2. satisfies $p_n \leq 2p_1$.

In the rest of the proof we assume 1. and 2.

(B) For a fixed set of periods $p_1, \ldots, p_n$, find a set $\mathcal{T}$ with periods $p_1, \ldots, p_n$ and utilization $U[p_1, \ldots, p_n]$.

(C) Show that $\min_{p_1,\ldots,p_n} U[p_1, \ldots p_n] = n(2^{1/n} - 1)$

Proof Sketch of Theorem 7

A set of tasks $\mathcal{T}$ *fully utilizes* the processor if it is schedulable by RM but any increase in execution time makes the set unschedulable.

Given $n \in \mathbb{N}$, we denote by b_n the greatest lower bound on utilization over all sets of n tasks that fully utilize the processor.

We prove that

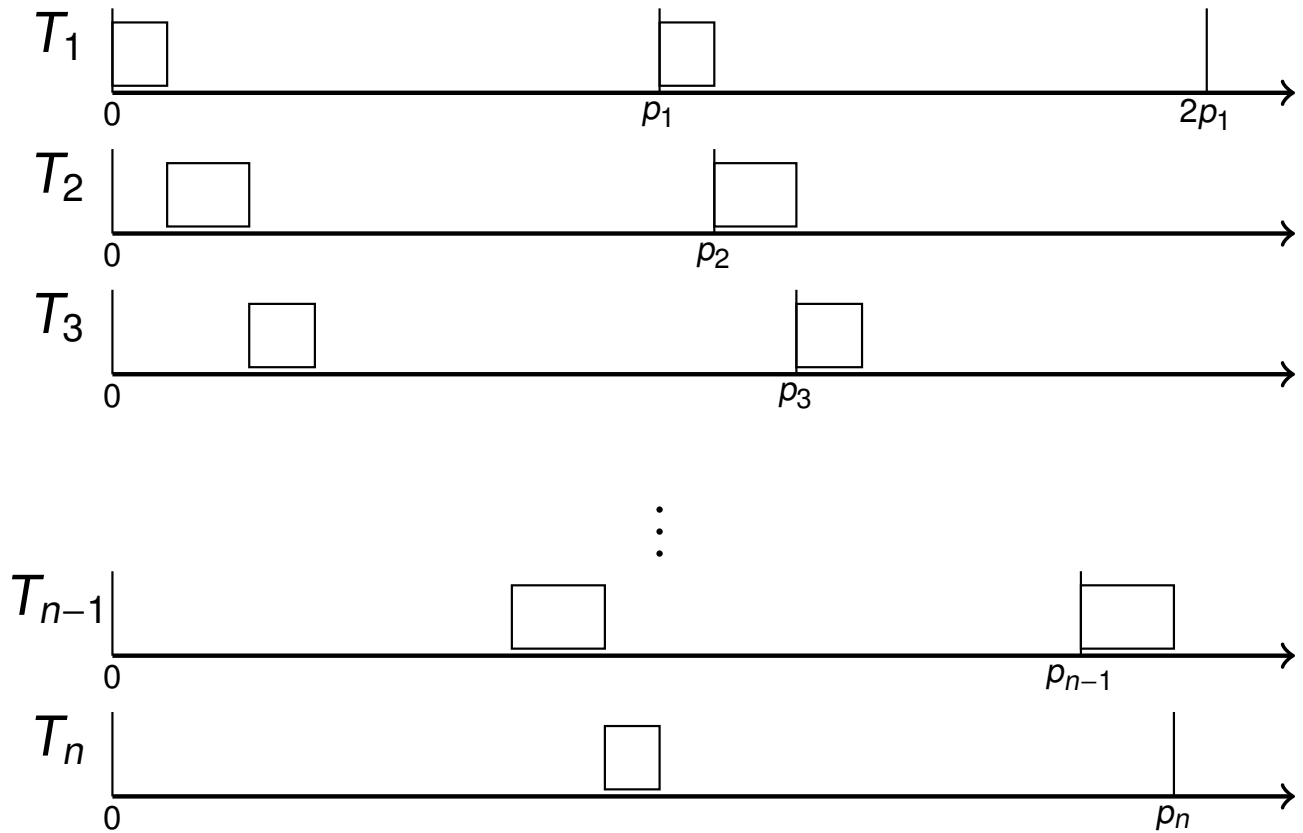
$$b_n = n(2^{\frac{1}{n}} - 1)$$

It immediately follows that for every $U > b_n$ there is a set of n tasks $\mathcal{T}$ that is not schedulable by *RM* but satisfies $U^{\mathcal{T}} \leq U$

We prove that if $U^{\mathcal{T}} \leq b_n$ for a given set $\mathcal{T}$ of n tasks, then $\mathcal{T}$ is schedulable using the RM algorithm by induction on n

Proof Sketch of Theorem 7 – Step (B)

In general, for $p_n \leq 2p_1$, the following instance gives b_n:



$$e_k = p_{k+1} - p_k \qquad \text{for } k = 1, \ldots, n-1$$

$$e_n = p_n - 2\sum_{k=1}^{n-1} e_k = 2p_1 - p_n$$

Time-Demand Analysis

Assume that $D_i \leq p_i$ for every i.

- Compute the total demand for processor time by a job released at a critical instant of a task, and by all the higher-priority tasks, as a function of time from the critical instant
- Check if this demand can be met before the deadline of the job:
  - Consider one task T_i at a time, starting with highest priority and working to lowest priority
  - Focus on a job $J_{i,c}$ in T_i, where the release time, t_0, of that job is a critical instant of T_i
  - At time $t_0 + t$ for $t \geq 0$, the processor time demand $w_i(t)$ for this job and all higher-priority jobs released in $[t_0, t]$ is bounded by

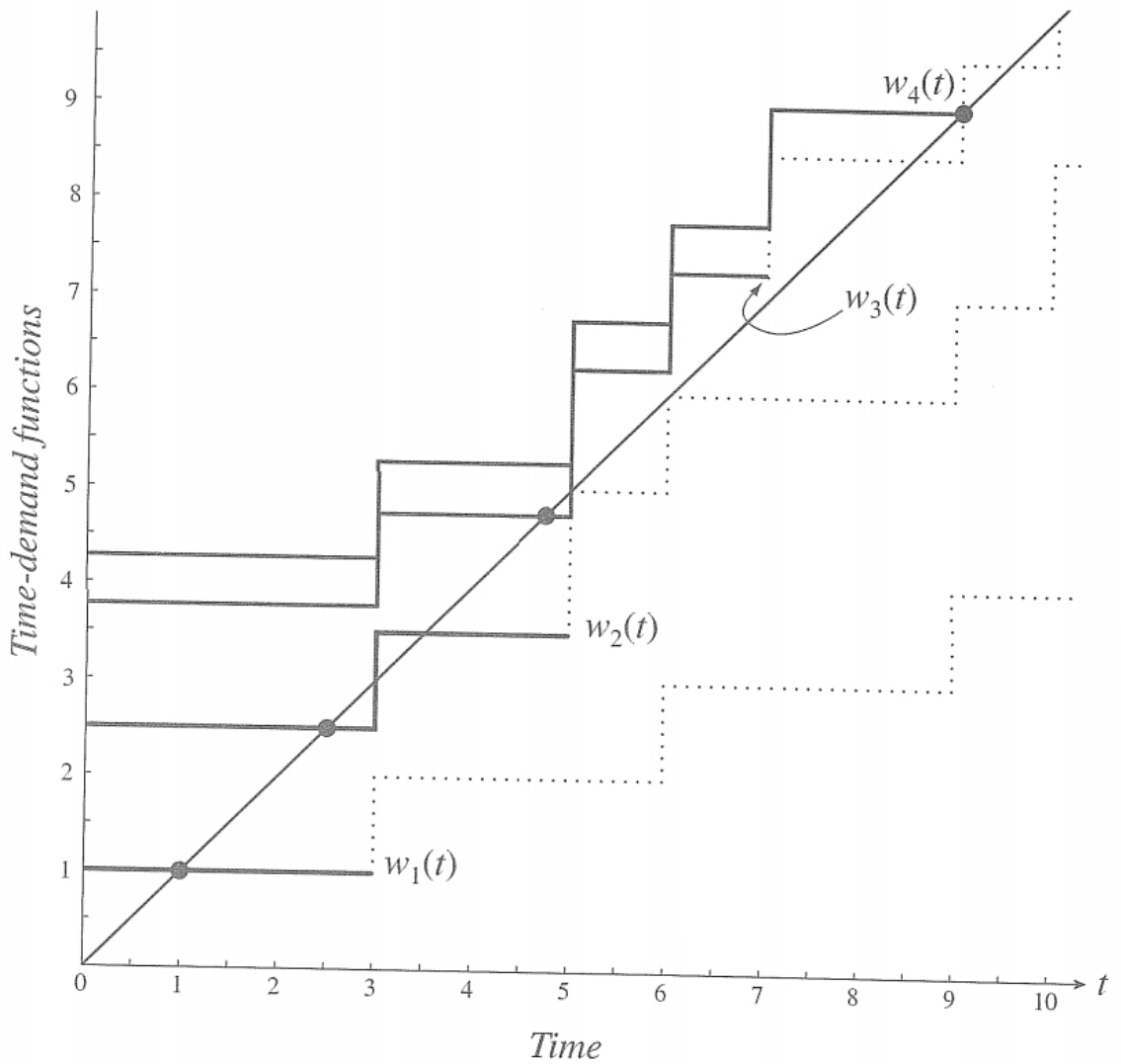
$$w_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil e_k \qquad \text{for } 0 < t \leq p_i$$

(Recall that by the critical instant theorem, the longest response time occurs when jobs of all tasks with higher priority are released at t_0)

Time-Demand Analysis

- Compare the time demand, $w_i(t)$, with the available time, t:
  - If $w_i(t) \leq t$ for some $t \leq D_i$, the job $J_{i,c}$ released at critical instant of T_i meets its deadline, $t_0 + D_i$
  - If $w_i(t) > t$ for all $0 < t \leq D_i$, then the task probably cannot complete by its deadline; and the system likely cannot be scheduled using a fixed priority algorithm
    (Note that this condition is only sufficient as the expression for $w_i(t)$ relies on the fact that jobs of all higher priority tasks are released at the critical instant t_0)
- Use this method to check that all tasks are schedulable if released at their critical instants; if so conclude the entire system can be scheduled

Time-Demand Analysis – Example



Example: $T_1 = (3, 1)$, $T_2 = (5, 1.5)$, $T_3 = (7, 1.25)$, $T_4 = (9, 0.5)$

This is schedulable by RM even though

$U^{\{T_1,\ldots,T_4\}} = 0.85 > 0.757 = U_{RM}(4)$

Time-Demand Analysis

- The time-demand function $w_i(t)$ is a staircase function
  - Steps in the time-demand for a task occur at multiples of the period for higher-priority tasks
  - The value of $w_i(t) - t$ linearly decreases from a step until the next step
- If our interest is the schedulability of a task, it suffices to check if $w_i(t) \leq t$ at the time instants when a higher-priority job is released and at D_i
- Our schedulability test becomes:
  - Compute $w_i(t)$
  - Check whether $w_i(t) \leq t$ for some t equal either to D_i, or to $j \cdot p_k$ where $k = 1, 2, \ldots, i$ and $j = 1, 2, \ldots, \lfloor D_i/p_k \rfloor$

Time-Demand Analysis

- Time-demand analysis schedulability test is more complex than the schedulable utilization test but more general:
  - Works for *any* fixed-priority scheduling algorithm, provided the tasks have short response time ($D_i \leq p_i$)
    Can be extended to tasks with arbitrary deadlines
- Still more efficient than exhaustive simulation
- Only a sufficient test (as well as the utilization test for fixed-priority systems)

Dynamic vs Fixed Priority

- EDF
  - pros:
    - optimal
    - very simple and complete test for schedulability
  - cons:
    - difficult to predict which job misses its deadline
    - strictly following EDF in case of overloads assigns higher priority to jobs that missed their deadlines
    - larger scheduling overhead
- DM (RM)
  - pros:
    - easier to predict which job misses its deadline (in particular, tasks are not blocked by lower priority tasks)
    - easy implementation with little scheduling overhead
    - (optimal in some cases often occurring in practice)
  - cons:
    - not optimal
    - incomplete and more involved tests for schedulability