Real-Time Scheduling

Scheduling of Reactive Systems

Priority-Driven Scheduling

Current Assumptions

- Single processor
- Fixed number, n, of *independent periodic* tasks

  i.e. there is no dependency relation among jobs
  - Jobs can be preempted at any time and never suspend themselves
  - No aperiodic and sporadic jobs
  - No resource contentions

Moreover, unless otherwise stated, we assume that

- **Scheduling decisions take place precisely at**
  - release of a job
  - completion of a job

  (and nowhere else)
- Context switch overhead is negligibly small

  i.e. assumed to be zero
- There is an unlimited number of priority levels

Fixed-Priority vs Dynamic-Priority Algorithms

A priority-driven scheduler is on-line
i.e. it does not precompute a schedule of the tasks

- It assigns priorities to jobs after they are released and places the jobs in a ready job queue in the priority order

  with the highest priority jobs at the head of the queue

- At each scheduling decision time, the scheduler updates the ready job queue and then schedules and executes the job at the head of the queue

  i.e. one of the jobs with the highest priority

Fixed-priority = *all jobs in a task* are assigned the same priority

Dynamic-priority = jobs in a task may be assigned different priorities

Note: In our case, a priority assigned to a job does not change. There are *job-level dynamic priority* algorithms that vary priorities of individual jobs – we won't consider such algorithms.

Fixed-priority Algorithms – Rate Monotonic

Best known fixed-priority algorithm is *rate monotonic (RM)* scheduling that assigns priorities to tasks based on their periods

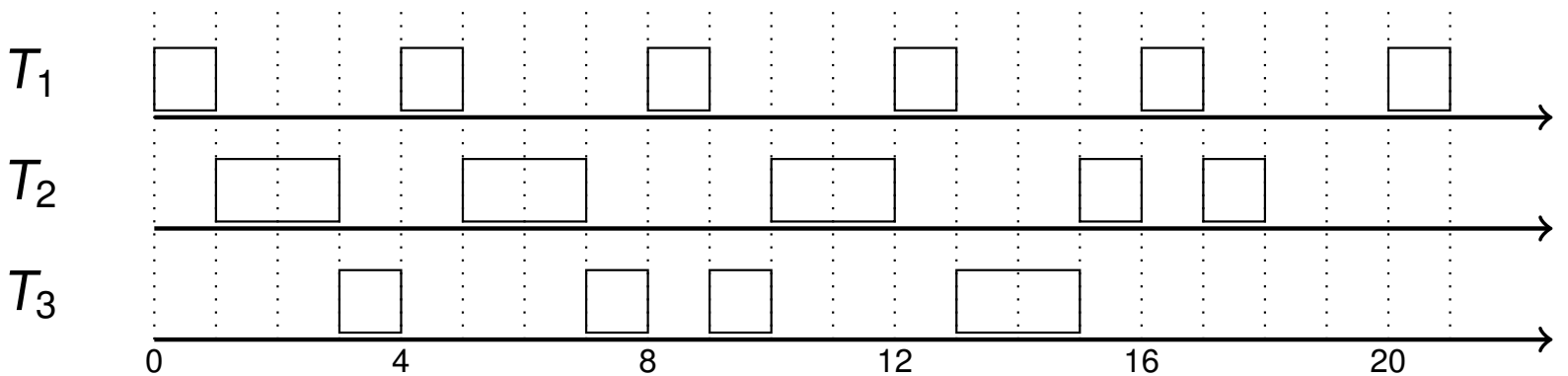
- The shorter the period, the higher the priority
- The *rate* is the inverse of the period, so jobs with higher rate have higher priority

RM is very widely studied and used

Example 13

$T_1 = (4, 1)$, $T_2 = (5, 2)$, $T_3 = (20, 5)$
with rates $1/4$, $1/5$, $1/20$, respectively

The priorities: $T_1 \succ T_2 \succ T_3$



Fixed-priority Algorithms – Deadline Monotonic

The *deadline monotonic (DM)* algorithm assigns priorities to tasks based on their *relative deadlines*

- the shorter the deadline, the higher the priority

Observation: When relative deadline of every task matches its period, then RM and DM give the same results

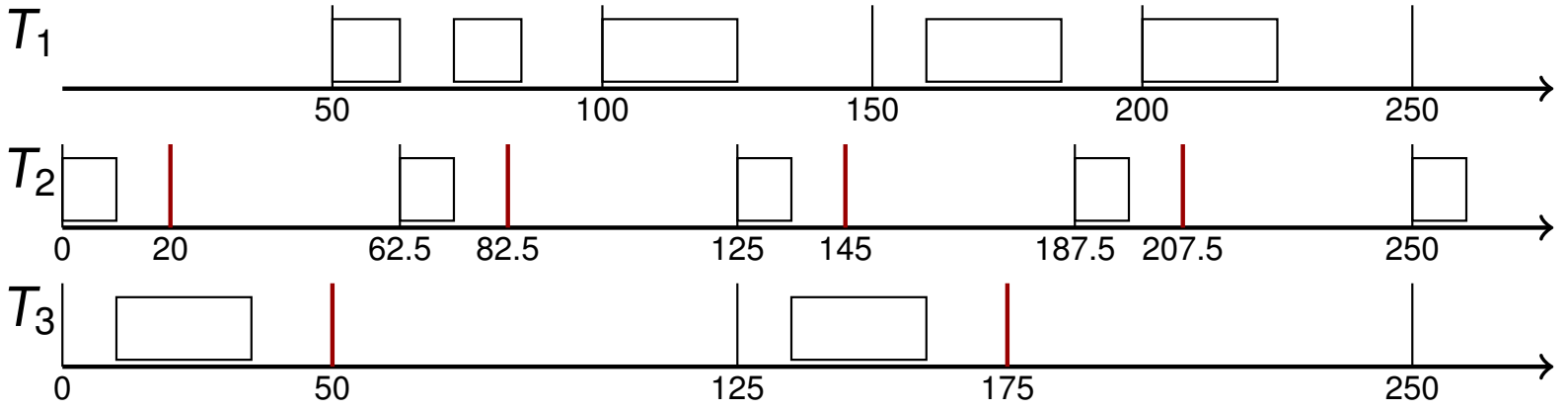
Proposition 1

When the relative deadlines are arbitrary DM can sometimes produce a feasible schedule in cases where RM cannot.

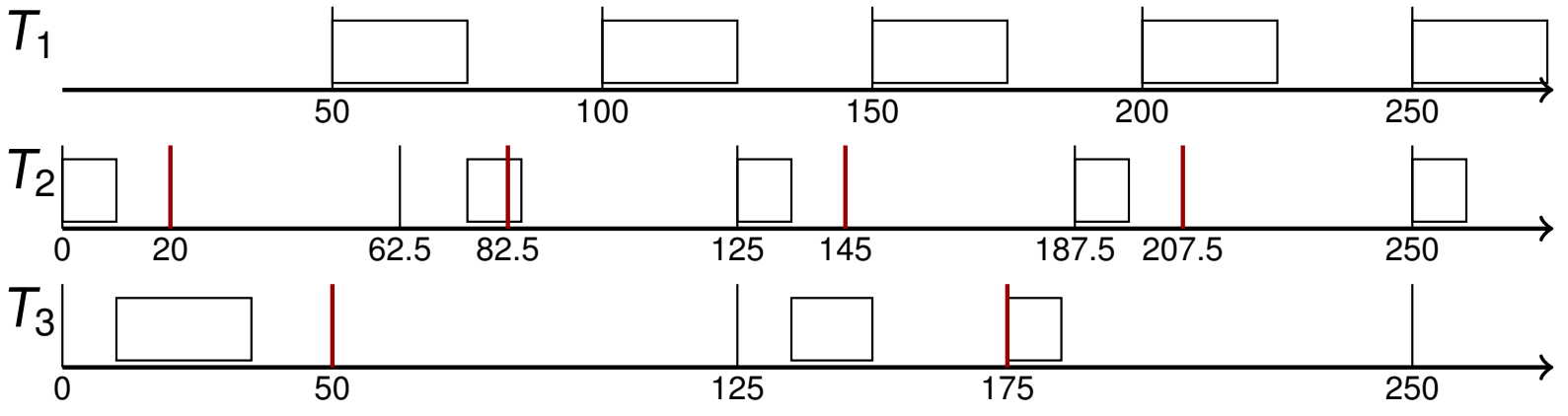
Rate Monotonic vs Deadline Monotonic

$T_1 = (50, 50, 25, 100)$, $T_2 = (0, 62.5, 10, 20)$, $T_3 = (0, 125, 25, 50)$

DM is optimal (with priorities $T_2 \succ T_3 \succ T_1$):



RM is not optimal (with priorities $T_1 \succ T_2 \succ T_3$):



Dynamic-priority Algorithms

Best known is *earliest deadline first (EDF)* that assigns priorities based on *current* (absolute) deadlines

- At the time of a scheduling decision, the job queue is ordered by earliest deadline

Another one is the *least slack time (LST)*

- The job queue is ordered by least slack time

Recall that the *slack time* of a job J_i at time t is equal to $d_i - t - x$ where x is the remaining computation time of J_i at time t

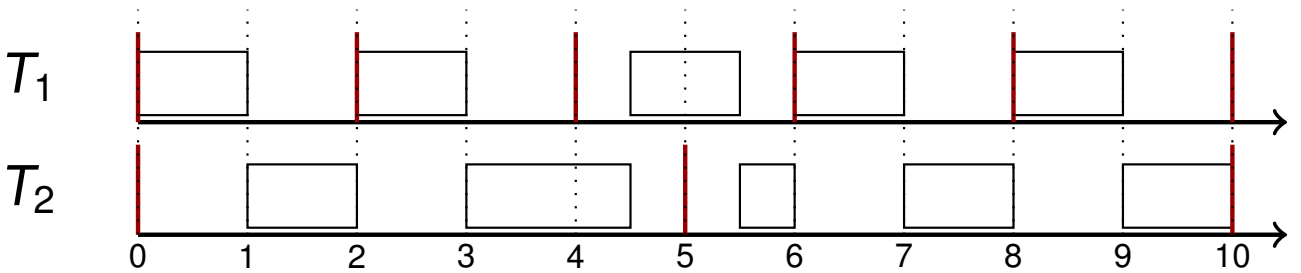
Comments:

- There is also a strict LST which reassigns priorities to jobs whenever their slacks change relative to each other – won't consider
- Standard "non real-time" algorithms such as FIFO and LIFO are also dynamic-priority algorithms

We focus on EDF here, leave some LST for homework

EDF – Example

$T_1 = (2, 1)$ and $T_2 = (5, 2.5)$



Note that the processor is 100% “utilized”, not surprising :-)

Summary of Priority-Driven Algorithms

We consider:
Dynamic-priority:

- **EDF** = at the time of a scheduling decision, the job queue is ordered by the earliest deadline

Fixed-priority:

- **RM** = assigns priorities to tasks based on their periods
- **DM** = assigns priorities to tasks based on their relative deadlines

(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?

To measure abilities of scheduling algorithms and to get fast online tests of schedulability we use a notion of **utilization**

Utilization

- *Utilization u_i of a periodic task T_i* with period p_i and execution time e_i is defined by $u_i := e_i/p_i$

  u_i is the fraction of time a periodic task with period p_i and execution time e_i keeps a processor busy
- *Total utilization $U^{\mathcal{T}}$ of a set of tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$* is defined as the sum of utilizations of all tasks of $\mathcal{T}$, i.e. by

$$U^{\mathcal{T}} := \sum_{i=1}^{n} u_i$$

- U is a *schedulable utilization* of an algorithm ALG if all sets of tasks $\mathcal{T}$ satisfying $U^{\mathcal{T}} \leq U$ are schedulable by ALG.

  Maximum schedulable utilization U_{ALG} of an algorithm ALG is the *supremum of schedulable utilizations of ALG*.

  - If $U^{\mathcal{T}} < U_{ALG}$, then $\mathcal{T}$ is schedulable by ALG.
  - If $U > U_{ALG}$, then there is $\mathcal{T}$ with $U^{\mathcal{T}} \leq U$ that is not schedulable by ALG.

Utilization – Example

- $T_1 = (2, 1)$ then $u_1 = \frac{1}{2}$
- $T_1 = (11, 5, 2, 4)$ then $u_1 = \frac{2}{5}$
  (i.e., the phase and deadline do not play any role)
- $\mathcal{T} = \{T_1, T_2, T_3\}$ where $T_1 = (2, 1)$, $T_2 = (6, 1)$, $T_3 = (8, 3)$ then

$$U^{\mathcal{T}} = \frac{1}{2} + \frac{1}{6} + \frac{3}{8} = \frac{25}{24}$$

Real-Time Scheduling

Priority-Driven Scheduling

Dynamic-Priority

Optimality of EDF

Theorem 14

Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a set of independent, preemptable periodic tasks with $D_i \geq p_i$ for $i = 1, \ldots, n$. The following statements are equivalent:

1. *$\mathcal{T}$ can be feasibly scheduled on one processor*
2. *$U^{\mathcal{T}} \leq 1$*
3. *$\mathcal{T}$ is schedulable using EDF*

(i.e., in particular, $U_{EDF} = 1$)

Proof.

1.⇒2. We prove that $U^{\mathcal{T}} > 1$ implies that $\mathcal{T}$ is not schedulable (whiteb.)

2.⇒3. Next slides and whiteboard ...

3.⇒1. Trivial

□

Proof of 2.⇒3. – Simplified

Let us start with a proof of a special case (see the assumptions A1 and A2 below). Then a complete proof will be presented.

We prove ¬3.⇒ ¬2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$.
(Note that the general case immediately follows.)

Assume that $\mathcal{T}$ is not schedulable according to EDF.
(Our goal is to show that $U^{\mathcal{T}} > 1$.)

This means that there must be at least one job that misses its deadline when EDF is used.

Simplifying assumptions:

A1 Suppose that all tasks are in phase, i.e. the phase $\varphi_\ell = 0$ for every task T_ℓ.

A2 Suppose that *the first job* $J_{i,1}$ of a task T_i misses its deadline.

By A1, $J_{i,1}$ is released at 0 and misses its deadline at p_i. Assume w.l.o.g. that this is the first time when a job misses its deadline.
(To simplify even further, you may (privately) assume that no other job has its deadline at p_i.)

Proof of 2.⇒3. – Simplified

Let G be the set of all jobs that are released in $[0, p_i]$ and have their deadlines in $[0, p_i]$.

Crucial observations:

- G contains $J_{i,1}$ and all jobs that preempt $J_{i,1}$.

  (If there are more jobs with deadline p_i, then these jobs do not have to preempt $J_{i,1}$. Assume the worst case: all these jobs preempt $J_{i,1}$.)

- The processor is never idle during $[0, p_i]$ and executes *only* jobs of G.

Denote by E_G the total execution time of G, that is, the sum of execution times of all jobs in G.

Corollary of the crucial observation: $E_G > p_i$ because otherwise $J_{i,1}$ (and all jobs that preempt it) would complete by p_i.

Let us compute E_G.

Proof of 2.⇒3. – Simplified

Since we assume $\varphi_\ell = 0$ for every T_ℓ, the first job of T_ℓ is released at 0, and thus $\left\lfloor \frac{p_i}{p_\ell} \right\rfloor$ jobs of T_ℓ belong to G.

E.g., if $p_\ell = 2$ and $p_i = 5$ then three jobs of T_ℓ are released in $[0, 5]$ (at times 0, 2, 4) but only $2 = \left\lfloor \frac{5}{2} \right\rfloor = \left\lfloor \frac{p_i}{p_\ell} \right\rfloor$ of them have their deadlines in $[0, p_i]$.

Thus the total execution time E_G of all jobs in G is

$$E_G = \sum_{\ell=1}^{n} \left\lfloor \frac{p_i}{p_\ell} \right\rfloor e_\ell$$

But then

$$p_i < E_G = \sum_{\ell=1}^{n} \left\lfloor \frac{p_i}{p_\ell} \right\rfloor e_\ell \leq \sum_{\ell=1}^{n} \frac{p_i}{p_\ell} e_\ell \leq p_i \sum_{\ell=1}^{n} u_\ell \leq p_i \cdot U^{\mathcal{T}}$$

which implies that $U^{\mathcal{T}} > 1$.

Proof of 2.⇒3. – Complete

Now let us drop the simplifying assumptions A1 and A2 !

Notation: Given a set of tasks $\mathcal{L}$, we denote by $\bigcup \mathcal{L}$ the set of all jobs of the tasks in $\mathcal{L}$.

We prove ¬3.⇒ ¬2. assuming that $D_i = p_i$ for $i = 1, \ldots, n$ (note that the general case immediately follows).

Assume that $\mathcal{T}$ is not schedulable by EDF. We show that $U^{\mathcal{T}} > 1$.

Suppose that a job $J_{i,k}$ of T_i misses its deadline at time $t = r_{i,k} + p_i$.
Assume that this is the earliest deadline miss.

Let $\mathcal{T}'$ be the set of all tasks whose jobs have deadlines (and thus also release times) in $[r_{i,k}, t]$
(i.e., a task belongs to $\mathcal{T}'$ iff at least one job of the task is released in $[r_{i,k}, t]$).

Let t_- be the end of the *latest* interval before t in which either jobs of $\bigcup(\mathcal{T} \setminus \mathcal{T}')$ are executed, or the processor is idle.

Then $r_{i,k} \geq t_-$ since all jobs of $\bigcup(\mathcal{T} \setminus \mathcal{T}')$ waiting for execution during $[r_{i,k}, t]$ have deadlines later than t (thus have lower priorities than $J_{i,k}$).

Proof of 2.⇒3. – Complete (cont.)

It follows that

- no job of $\bigcup(\mathcal{T} \setminus \mathcal{T}')$ is executed in $[t_-, t]$,
  (by definition of t_-)
- the processor is fully utilized in $[t_-, t]$.
  (by definition of t_-)
- *all jobs* (that all must belong to $\bigcup \mathcal{T}'$) *executed in* $[t_-, t]$ *are released in* $[t_-, t]$ *and have their deadlines in* $[t_-, t]$ since
  - no job of $\bigcup \mathcal{T}'$ executes just before t_-,
  - all jobs of $\bigcup \mathcal{T}'$ released in $[t_-, r_{i,k}]$ have deadlines before t,
  - jobs of $\bigcup \mathcal{T}'$ released in $[r_{i,k}, t]$ with deadlines after t are not executed in $[r_{i,k}, t]$ as they have lower priorities than $J_{i,k}$.

Let G be the set of all jobs that are released in $[t_-, t]$ and have their deadlines in $[t_-, t]$.

Note that $J_{i,k} \in G$ since $r_{i,k} \geq t_-$.

Denote by E_G the sum of all execution times of all jobs in G (the total execution time of G).

Proof of 2.⇒3. – Complete (cont.)

Now $E_G > t - t_-$ because otherwise $J_{i,k}$ would complete in $[t_-, t]$.

How to compute E_G?

For $T_\ell \in \mathcal{T}'$, denote by R_ℓ the earliest release time of a job in T_ℓ during the interval $[t_-, t]$.

For every $T_\ell \in \mathcal{T}'$, exactly $\left\lfloor \frac{t-R_\ell}{p_\ell} \right\rfloor$ jobs of T_ℓ belong to G. (For every $T_\ell \in \mathcal{T} \setminus \mathcal{T}'$, exactly 0 jobs belong to G.)

Thus

$$E_G = \sum_{T_\ell \in \mathcal{T}'} \left\lfloor \frac{t - R_\ell}{p_\ell} \right\rfloor e_\ell$$

As argued above:

$$t-t_- < E_G = \sum_{T_\ell \in \mathcal{T}'} \left\lfloor \frac{t - R_\ell}{p_\ell} \right\rfloor e_\ell \leq \sum_{T_\ell \in \mathcal{T}'} \frac{t - t_-}{p_\ell} e_\ell \leq (t-t_-) \sum_{T_\ell \in \mathcal{T}'} u_\ell \leq (t-t_-) U^{\mathcal{T}}$$

which implies that $U^{\mathcal{T}} > 1$.

Density and EDF

What about tasks with $D_i < p_i$?

Density of a task T_i with period p_i, execution time e_i and relative deadline D_i is defined by

$$e_i / \min(D_i, p_i)$$

Total density $\Delta^{\mathcal{T}}$ of a set of tasks $\mathcal{T}$ is the sum of densities of tasks in $\mathcal{T}$

Note that if $D_i < p_i$ for some i, then $\Delta^{\mathcal{T}} > U^{\mathcal{T}}$

Theorem 15

A set $\mathcal{T}$ of independent, preemptable, periodic tasks can be feasibly scheduled on one processor if $\Delta^{\mathcal{T}} \leq 1$.

Note that this is NOT a necessary condition! (Example whiteb.)

Schedulability Test For EDF

The problem: Given a set of independent, preemptable, periodic tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ where each T_i has a period p_i, execution time e_i, and relative deadline D_i, decide whether $\mathcal{T}$ is schedulable by EDF.

Solution using utilization and density:

If $p_i \leq D_i$ for each i, then it suffices to decide whether $U^{\mathcal{T}} \leq 1$.

Otherwise, decide whether $\Delta^{\mathcal{T}} \leq 1$:

- If yes, then $\mathcal{T}$ is schedulable with EDF
- If not, then $\mathcal{T}$ does not have to be schedulable

Note that

- Phases of tasks do not have to be specified
- Parameters may vary: increasing periods or deadlines, or decreasing execution times does not prevent schedulability

Schedulability Test for EDF – Example

Consider a digital robot controller

- A control-law computation
  - takes no more than 8 ms
  - the sampling rate: 100 Hz, i.e. computes every 10 ms

  Feasible? Trivially yes ....
- Add Built-In Self-Test (BIST)
  - maximum execution time 50 ms
  - want a minimal period that is feasible (max one second)

  With 250 ms still feasible ....
- Add a telemetry task
  - maximum execution time 15 ms
  - want to minimize the deadline on telemetry
    period may be large

  Reducing BIST to once a second, deadline on telemetry may be set to 100 ms ....