Advanced Real-Time Systems

Lecture 1/6

Enrico Bini

October 25, 2012

Outline

Course for PhD:

- Trying to give a broad view of the research area at large;
- Expose quickly to many topics: you better slow me down by asking question!
- Much work/study has to be made at home;
- When no reference is provided, wikipedia is fine.

Examination:

- homework assigned on Friday to be completed on by Thursday morning

Material:

- union slides plus notes (check the website)

Outline

Basic ingredients

Basic elements of a scheduling problem:

- a set $\mathcal{N}$ of *tasks* (aka demands, works, jobs) requiring work to be made. Since they are finite, we represent them by $\mathcal{N} = \{1, 2, \ldots, n\}$;
- a set $\mathcal{R}$ of *resources* (aka processors, machines, workers, etc.) capable to perform some work (one/many machine, heterogeneous multicore, different machines in manufacturing, etc.). Since they are finite we represent them by $\mathcal{R} = \{1, 2, \ldots, m\}$;
- a *time set* $\mathcal{T}$, over which the scheduling is performed (typically $\mathbb{N}$ or $[0, \infty)$);
- a *scheduling algorithm* $\mathcal{A}$ which produces a *schedule* S for given $\mathcal{R}$ and $\mathcal{N}$.

Task

Characteristics of tasks:

- amount of work,
- recurrent/non-recurrent, does a task repeat over time? How often?
- on-line/off-line, do we know the parameters in advance?
- precedence constraints,
- deadlines, "the work must be completed by this instant",
- affinity to resources, not all resources are the same,
- sequential (only one resource at time), parallel (more than one resource at time), parallelizable (one or more resources at time).

Resource

Characteristics of a resource:

- type, (coffee machine $\neq$ a CPU)
- execution rate r_k (speed), which may be time-varying or demand-varying;
- operating modes (variable speed over time, etc.)

Schedule

A schedule is a function

$$S : \mathcal{R} \times \mathcal{T} \to \mathcal{N} \cup \{0\}$$

If $m = 1$ then just $S : \mathcal{T} \to \mathcal{N} \cup \{0\}$.

- If $S(k, t) = i$ then the resource k is assigned to the i-task at time t.
- If $S(k, t) = 0$ then the resource k is not assigned at time t (we say that the k-th resource is *idle* at t).

- This definition of schedule implies that at every instant t each resource is assigned to at most one task.
- Conversely, at every instant each task may be assigned any number of resources in $\mathcal{R}$.

Viewing a schedule

The inverse image of i under S, $s_i \subseteq \mathcal{R} \times \mathcal{T}$, that is

$$s_i = S^{-1}(i) = \{(k,t) \in \mathcal{R} \times \mathcal{T} : S(k,t) = i\}$$

represents the resources allocated to the i-th task.

- Draw a task schedule

Scheduling algorithms

Goal of scheduling algorithm $\mathcal{A}$: find a schedule S such that:

- the constraints are met (in this case constraints have to be specified): all task deadlines are met,
- some target function is minimized/maximized (minimum makespan/delay, best "performance": requires to know how the timing affect the "performance")

Characteristics of scheduling algorithms:

- (non-)work-conserving: no idle resource if pending tasks exist;
- (non-)preemptive, I can interrupt a task while it executes;
- time-complexity: how long does it take to decide the resource assignment?

Examples of scheduling algorithms

Examples of scheduling algorithms:

- First In First Out (FIFO), schedule tasks in order of arrivals;
- Round Robin (RR), divide the time in slices and assign slices in round;
- Shortest Job First (SJF) and its preemptive version Shortest Remaining Time First (SRTF);
- Earliest Deadline First (EDF), assigns priority according to the deadlines d;
- Least Laxity First (LLF), aka Least Slack Time (LST), at t assigns priority according to the smallest "laxity" $(d - t) - c'$
- Fixed Priorities (FP), tasks are prioritized.

Feasibility vs. Schedulability

Definition

A task set $\mathcal{N}$ is *feasible* is it exists a schedule which satisfies the task constraint.

Definition

A task set $\mathcal{N}$ is *schedulable* by the scheduling algorithm $\mathcal{A}$, if $\mathcal{A}$ can produce a schedule S which does not violate any constraint of $\mathcal{N}$.

Obviously: schedulability by any algorithm implies feasibility.

Outline

RT Task Model

- Task i often denoted by τ_i
- Tasks are recurrent and activated *sporadically*: with a minimum interarrival (or *period*) T_i;
- At each activation of a task it is required the execution of a *job* (job $\neq$ task);
- All jobs belonging to τ_i have an execution requirement C_i;
- All jobs belonging to τ_i have a *relative deadline* D_i, relative to the activation
  - if $D_i = T_i$ then *implicit deadline*,
  - if $D_i \leq T_i$ then *constrained deadline*,
  - if D_i unrelated to T_i then *arbitrary deadline*.

Also the quantity $U_i = C_i/T_i$ is called *task utilization* and represents the fraction of time needed by task i.

Resource Model

- Resources: single processor, multiprocessor (with m processors/cores).
- A necessary condition for feasibility is:

$$\sum_{i=}^{n} U_i \leq m \tag{1}$$

In the course, we will focus on single processor only ($m = 1$).

Outline

Priorities

- Tasks are sorted in decreasing priority order
  - τ_1 is the highest priority one,
  - τ_n is the lowest priority one.
- Draw an example of how FP schedule tasks.
- What is the best priority assignment?

Optimal priority assignment

Theorem (Liu, Layland, 1973 [?])

If $D_i = T_i$ then Rate Monotonic (RM) is **optimal***: if some priority assignment can schedule the task set, then RM can schedule the task set.*

Theorem (Leung, Whitehead, 1982 [?])

If $D_i \leq T_i$ then Deadline Monotonic (DM) is optimal.

Utilization upper bound

Liu and Layland [**?**] also proved the most popular *utilization upper bound* (checked on 24/10/2012: cited 7914 in Google Scholar).

Theorem

If $D_i = T_i$

$$\sum_{i=1}^{n} U_i \leq n(\sqrt[n]{2} - 1)$$

then $\mathcal{T}$ *is schedulable by RM.*

- The RHS is called *utilization upper bound*.
- As $n \to \infty$ the bound tends to $\log 2 \approx 0.69315$
- Is the LL bound tight? Is there any non-schedulable task set with $\sum_i U_i > U_{\mathsf{LL}}$? Draw the example.

Hyperbolic Bound

Theorem (Hyperbolic Bound [?])

If $D_i = T_i$ and

$$\prod_{i=1}^{n}(1 + U_i) \leq 2$$

then $\mathcal{T}$ is schedulable by RM.

- Visualization of the bound and interpretation of HB in the utilization space.
- Still some space for uncertainty. What happens in between?

Outline

Task Response time

Definition

The *response time* R_i of task τ_i is the longest time that can elapse from the activation of any job to its completion.

$$R_i = \max_{j \geq 1} \{R_{i,j}\}$$

with $R_{i,j}$ response time of the j-th job of τ_i.

The idea: to compute the response time and check whether or not $R_i \leq D_i$

- if so, then all jobs of τ_1 will meet their deadline (schedulable by FP).
- if not, then some job will miss its deadline (not schedulable by FP).

Computing R_i: interference

Definition

We define the *level-i interference* $I_i(t)$ as the maximum amount of work which can be requested by tasks with priority higher than i in an interval of length t.

For our simple task model, it is

$$I_i(t) = \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

and it corresponds to the scenario with all tasks activated together at 0 at the highest possible rate.

- Example of interference with different activation patterns (two alternating periods)

Computing R_i: recurrent equation

- The response time of the first job of τ_i is found as the smallest fixed point of the following equation

$$\begin{cases} R_{i,1}^{(0)} = C_i \\ R_{i,1}^{(k+1)} = C_i + I_i(R_{i,1}^{(k)}) \end{cases} \tag{2}$$

- It converges iff $\sum_{j=1}^{i-1} U_j < 1$
- Explain its rationale.

Arbitrary deadline case

- if $R_{i,1} \leq T_i$ then $R_i = R_{i,1}$ is the longest response time among all jobs belonging to τ_i.
- otherwise $(R_{i,1} > T_i)$ it is not guaranteed that the maximum job response time occurs at the first job! In such a case we have to compute the response time $R_{i,k}$ of all subsequent jobs.
- We care only if $D_i > T_i$ (arbitrary deadline)
- We can compute the *absolute job response time* $r_{i,j}$ as follows

$$\begin{cases} r_{i,1} = R_{i,1} \\ r_{i,j}^{(0)} = r_{i,j-1} + C_i \\ r_{i,j}^{(k+1)} = j\,C_i + I_i(r_{i,j}^{(k)}) \\ R_{i,j} = r_{i,j} - (j-1)T_i \end{cases} \tag{3}$$

until $r_{i,j} \leq j\,T_i$.
- The interval $[0, r_{i,j^*}]$, with j^* equal to the index of job where it first is $r_{i,j} \leq j\,T_i$, is called *level-i busy period*.

Arbitrary deadline case

- If $\sum_{j=1}^{i} U_j < 1$, j^* is finite
- If $\sum_{j=1}^{i} U_j > 1$, $\lim_j R_{i,j} = \infty$ (*level-i overload*)
- If $\sum_{j=1}^{i} U_j = 1$ and $\{T_1, \ldots, T_i\}$ rational, j^* finite because the schedule will repeat after the least common multiple of the periods
- If $\sum_{j=1}^{i} U_j = 1$ and $\{T_1, \ldots, T_i\}$ irrational, j^* infinite, but R_i can still be defined as $R_i = \sup_j R_{i,j}$
- In human cases $R_i = \max_{j \leq j^*} R_{i,j}$ and we can check $R_i \leq D_i$.

Response time upper bound

- Iterating the response time equation Eq. (3) may be too time consuming, especially if it has to be executed on-line.
- One may want to forget necessity by computing a response time upper bound.
- Suppose we have a linear upper bound of the interference $I_i(t) \leq \overline{I}_i(t) = \alpha_i t + \beta_i$. Then from (3)

$$
\begin{aligned}
R_i &\leq C_i + \overline{I}_i(R_i) = C_i + \alpha_i R_i + \beta_i \\
R_i - \alpha_i R_i &\leq C_i + \beta_i \\
R_i &\leq \frac{C_i + \beta_i}{1 - \alpha_i} = \overline{R}_i
\end{aligned}
$$

and have the following as a just sufficient (faster) test

$$\forall i, \quad \overline{R}_i \leq D_i$$

Resp. time up. bound: coefficients

Finding suitable α_i and β_i. By upper bounding the $\lceil x \rceil$ with $x+1$ in $I_i(t)$ we quickly find

$$\alpha_i = \sum_{j=1}^{i-1} U_j \qquad \beta_i = \sum_{j=1}^{i-1} C_j$$

however β_i can be made [**?**] a bit tighter by chosing

$$\beta_i = \sum_{j=1}^{i-1} C_j(1-U_j)$$

this bound is still valid in the arbitrary deadline case.