Real-Time Systems (for PhD students)

Enrico Bini

University of Turin

2024, Unito

Outline

1. Introduction to the course

2. Scheduling Problems

Presentation

- Course for PhD: evolving, open problems, there may be mistakes, it requires interaction
- Goal: to give a broad view of the real-time systems area at large
- Lecture style: expose quickly many topics ⇒ you better slow me down by asking question!
- Schedule of classes (subscribe to the Google Calendar)

Tue	Apr 09	15:00–17:00	Sala Sem. Primo piano
Fri	Apr 12	14:00–16:30	Sala Sem. Primo piano
Tue	Apr 23	14:30–17:00	Sala Riun. Primo piano
Fri	Apr 24	14:30–17:00	Sala Riun. Primo piano
...	...	...	...

- Evaluation:
  - a homework: about one week of time will be given
  - the presentation of an assigned research paper
- On Tuesday, April 23, each student **presents his/her research area**
  - for brainstorming about the to-be-chosen paper to study

Behaviour of computing systems

- Logic gates (AND, OR, NOT. . .) have a deterministic behavior
  - (actually, new integration technologies are challenging even the behavior of basic logic gates)
- The exact model of a computing system by analyzing at the level of logic gates is infeasible
- Exactness is forgotten and **uncertainties** are introduces
- Sources of uncertainties
  1. input: what data will a sensor read?
  2. state: what will be the state (registers, cache, memory) when executing some code?

Model of uncertainties

Different approaches to uncertainties

- Benchmarking: “we try a couple of references applications to imagine the behaviour in general”
  - pros: easy to implement, the only feasible on general purpose computing
  - cons: what can we really say in general?
- Stochastic models: “we attach a probability measure onto the uncertainties”
  - pros: analytic results are possible via (stochastic) formal methods, queueing theory, etc.
  - cons: what is the probability measure of the input? What about the cache content?
- Bounding uncertainties with intervals, polytopes, etc.
  - pros: analytic results are possible via (deterministic) formal methods, real-time scheduling, etc.
  - cons: results may be very conservative as they may be dominated by corner cases which do not appear in reality

Outline

1. Introduction to the course

2. Scheduling Problems

Basic ingredients of scheduling

- a *time set* $\mathcal{T}$ (typically $\mathbb{N}$ or $[0, \infty)$);
- 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 $\mathcal{M}$ of *machines* (aka processors, resources, workers, etc.) capable to perform some work (one/many machine, heterogeneous multicore, etc.). Since they are finite we represent them by $\mathcal{M} = \{1, 2, \ldots, m\}$;
- a set $\mathcal{C}$ of *constraints*, such as:
  1. affinities: "task i may execute only over $\mathcal{M}_i \subseteq \mathcal{M}$
  2. deadlines: "task i must complete not later than a deadline D"
  3. mutual exclusion: "task i may not execute while task j is executing"
  4. parallelism of tasks: "task i needs 3 processors in parallel when executing
- a *scheduling algorithm* $\mathcal{A}$ which produces a *schedule* S of $\mathcal{N}$ over $\mathcal{M}$, compatible with $\mathcal{C}$.

Task 1/2

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?
- sequential (only one machine at time), parallel (more than one machine at time), parallelizable (one or more machines at time).

Task 2/2

- we denote the *work request function* of the i task by $r_i(t)$. If the work is composed by a sequence of jobs of sizes $\{C_{i,j}\}$, requested at instants $\{a_{i,j}\}$, then
$$r_i(t) = \text{draw step function of size } C_{i,j} \text{ at time } a_{i,j}.$$
- we denote the *pending work* of the i task by $w_i(t)$

Basic principle

$$\forall t_0, t_1 \geq t_0, \quad w_i(t_1) \leq w_i(t_0) + r_i(t_1) - r_i(t_0)$$

Machine

Characteristics of a machine:

- type, (CPU, GPU, etc.)
- speed $\sigma_k(t)$ of machine k at time t, which may be time-varying;
- operating modes (variable speed over time, etc.)
- availability of dedicated resources (printer, some dedicated hardware, etc.)

Constraints

- precedence constraints: "this task can start only after this other task"
- deadlines: "the work must be completed by this instant",
- affinity to machines "this task must always run over this subset of machines"
- mutual exclusion, "these portions of tasks cannot be scheduled at the same time"

Schedule

A schedule is a function

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

with "task 0" denoting *idle*, compatible with the constraints in $\mathcal{C}$.
If $m = |\mathcal{M}| = 1$ (one machine) then just $S : \mathcal{T} \to \mathcal{N} \cup \{0\}$.

- If $S(k,t) = i$ then the machine k is assigned to the i-task at time t.
- If $S(k,t) = 0$ then the machine k is not assigned at time t (we say that the k-th machine is *idle* at t).
- This definition of schedule implies that at every instant t each machine is assigned to at most one task.
- Conversely, at every instant each task may be assigned any number of machines in $\mathcal{M}$. (possible in case of task parallelism)

Example of schedule

- The inverse image of i under S, $S^{-1}(i) \subseteq \mathcal{M} \times \mathcal{T}$, that is

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

  represents the machines allocated to the i-th task.
- Draw a task schedule
- the i-th task is *sequential* if

$$\forall (k,t),(k',t') \in S^{-1}(i),\ t = t', \quad \Rightarrow \quad k = k'$$

  then we can define $s_i(t)$ indicator function of
- a schedule S is *partitioned* if

$$\forall i \neq 0, \forall (k,t),(k',t') \in S^{-1}(i), \quad k = k'$$

- task i has *affinity* $\mathcal{M}_i \subseteq \mathcal{M}$ if

$$\forall (k,t) \in S^{-1}(i), \quad k \in \mathcal{M}_i$$

Conservation of work

Law of "conservation of work"

$$\forall t_0 < t_1, \quad w_i(t_1) = w_i(t_0) - s_i(t_0, t_1) + \overbrace{r_i(t_1) - r_i(t_0)}^{\text{work requested in } [t_0, t_1)}$$

with the *scheduled resource* $s_i(t_0, t_1)$ to the i task, defined by

$$s_i(t_0, t_1) = \int_{t_0}^{t_1} \sum_{(k,t) \in S^{-1}(i)} \sigma_k(t)\, dt$$

with $\sigma_k(t)$, speed of k-th machine, at time t. If machines are identical then $\sigma_k = 1$

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). Example: 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:

- *work-conserving*: no idle machine if pending tasks exist;

$$\forall k \in \mathcal{M},\ S(k,t) = 0 \quad \Rightarrow \quad \forall i \in \mathcal{N}, w_i(t) = 0$$

- *preemptive*, $\mathcal{A}$ can interrupt a task while it executes;
- time-complexity: how long does it take to decide the machine assignment?
- clairvoyant: takes decisions based on the knowledge of the future

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" $(d - t) - w(t)$
- 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 $\mathcal{C}$.

- Goal of scheduling algorithm design: to design a scheduling algorithm $\mathcal{A}$ which can schedule any feasible task set.

Analysis, sensitivity, and design

- Given a scheduling algorithm $\mathcal{A}$, we distinguish three problems (of increasing difficulty):
  1. **analysis**: given a set $\mathcal{N}$ of tasks and a set $\mathcal{M}$ of machines, is $\mathcal{N}$ schedulable by $\mathcal{A}$ over $\mathcal{M}$?
  2. **sensitivity**: given a set $\mathcal{N}$ of tasks schedulable by $\mathcal{A}$ over $\mathcal{M}$. How much can we modify $\mathcal{N}$ such that it remains schedulable?
  3. **optimal design**: given a set $\mathcal{N}$ of tasks. What is the best set of machines $\mathcal{M}$ such that $\mathcal{N}$ is schedulable by $\mathcal{A}$ over $\mathcal{M}$?