Global Scheduling in Multiprocessor Real-Time Systems

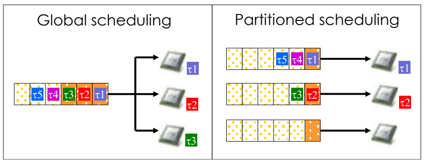
Alessandra Melani

Scuola Superiore Sant'Anna – Institute of Communication, Information and Perception Technologies

Retis Real-Time Systems Laboratory

Global vs Partitioned scheduling

- Single shared queue instead of multiple dedicated queues



Bin-packing problem + Uniprocessor scheduling problem

NP-hard in the strong sense; various heuristics adopted

Well-known

Pros and cons

- **Global** scheduling
  - ✓ Automatic load balancing
  - ✓ Lower avg. response time
  - ✓ Simpler implementation
  - ✓ *Optimal* schedulers exist
  - ✓ More efficient reclaiming
  - ✗ Migration costs
  - ✗ Inter-core synchronization
  - ✗ Loss of cache affinity
  - ✗ Weak scheduling framework
- **Partitioned** scheduling
  - ✓ Supported by automotive industry (e.g., AUTOSAR)
  - ✓ No migrations
  - ✓ Isolation between cores
  - ✓ Mature scheduling framework
  - ✗ Cannot exploit unused capacity
  - ✗ Rescheduling not convenient
  - ✗ NP-hard allocation

Main (negative) results

- **Weak theoretical framework** ☹
  - Unknown critical instant
  - G-EDF is not optimal
  - Any G-JLFP scheduler is not optimal
  - Optimality only for implicit deadlines
  - Many sufficient tests (most of them incomparable)

Unknown critical instant

- **Critical instant**
  - Job release time such that response-time is maximized
- **Uniprocessor**
  - Liu & Layland: synchronous release sequence yields worst-case response-times
    - Synchronous: all tasks release a job at time 0
    - Assuming constrained deadlines and no deadline misses
- **Multiprocessors**
  - No general critical instant is known!
  - It is not necessarily the synchronous release sequence…

Unknown critical instant

- Synchronous periodic arrival of jobs is not a critical instant for multiprocessors

(C_i, D_i, T_i)

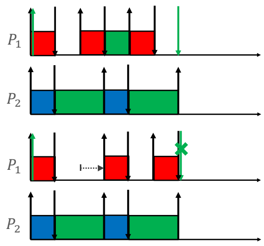
$\tau_1 = (1, 1, 2)$

$\tau_2 = (1, 1, 3)$

$\tau_3 = (5, 6, 6)$

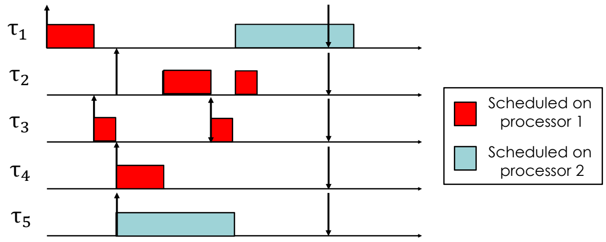
Synchronous periodic situation

The second job of τ_1 is delayed by one unit



We need to find pessimistic situations to derive *sufficient* schedulability tests

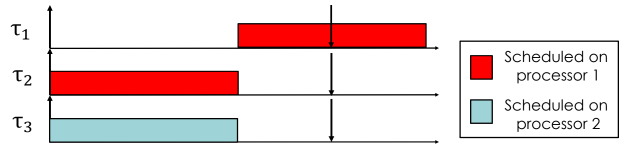
G-EDF is not optimal



- **Uniprocessors**
  - EDF is optimal
- **Multiprocessors**
  - G-EDF is not optimal
  - Key problem: **sequentiality** of tasks
  - Two processors available for τ_1, but it can only use one

Any G-JLFP scheduler is not optimal

Two processors, three tasks, $T_i = 15$, $C_i = 10$



- **Any job-level fixed-priority scheduler is not optimal**
  - Synchronous release time
  - One of the three jobs is scheduled last under any JLFP policy
  - Deadline miss unavoidable!

G-JLDP required for optimality

- **G-JLDP:** *Global Job Level Dynamic Priority*; the priority of each job may change over time

Proportionate fairness

- P-fair: notion of "fair share of processor"
- If a schedule is P-fair, no **implicit** deadline will be missed → **optimal** algorithm

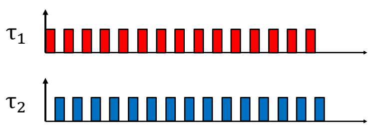
Basic principle:

- Timeline is divided into **equal length slots**
- Task period and execution time are multiples of the slot size
- Each task receives amount of slots **proportional to its task utilization**
  - If a task has utilization $U = \frac{C_i}{T_i}$, then it will have been allocated $U * t$ time slots for execution in the interval $[0, t]$

Proportionate fairness

Example:

- $C_1 = C_2 = 3$; $T_1 = T_2 = 6$ $(U_1 = U_2 = \frac{1}{2})$



- Quantum-based: $C_i \in \mathbb{Z}^+, T_i \in \mathbb{Z}^+$; scheduling decisions can only occur at integers
- A task executes during a whole time slot or not execute at all in that time slot

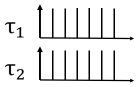
Proportionate fairness

$$\underbrace{lag(\tau_i, t)}_{\text{Error}} = \underbrace{t * \left(\frac{C_i}{T_i}\right)}_{\text{"Fluid" execution: should have executed in } [0,t)} - \underbrace{allocated(\tau_i, t)}_{\text{Real execution in } [0,t)}$$

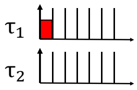
- **Goal:** find an algorithm that minimizes $\max_t |lag(\tau_i, t)|$
- Which are the values that $lag(\tau_i)$ can take?

Proportionate fairness

- Example: $\tau = \{(T_1 = 5,\ C_1 = 2),\ (T_2 = 7,\ C_2 = 4)\}$, 1 processor

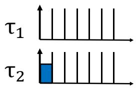


No task executes in [0,1)
$lag(\tau_1, 1) = 1 * \left(\frac{2}{5}\right) - 0 \neq 0$
$lag(\tau_2, 1) = 1 * \left(\frac{4}{7}\right) - 0 \neq 0$



Task τ_1 executes in [0,1)
$lag(\tau_1, 1) = 1 * \left(\frac{2}{5}\right) - 1 \neq 0$
$lag(\tau_2, 1) = 1 * \left(\frac{4}{7}\right) - 0 \neq 0$

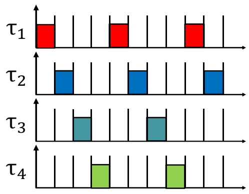
$lag(\tau_i, 1) = 0$ is impossible at time 1



Task τ_2 executes in [0,1)
$lag(\tau_1, 1) = 1 * \left(\frac{2}{5}\right) - 0 \neq 0$
$lag(\tau_2, 1) = 1 * \left(\frac{4}{7}\right) - 1 \neq 0$

Proportionate fairness

- Example: $\tau = \{(T_1 = 4,\ C_1 = 1),\ (T_2 = 4,\ C_2 = 1), (T_3 = 4,\ C_3 = 1), (T_2 = 4,\ C_2 = 1)\}$, one processor

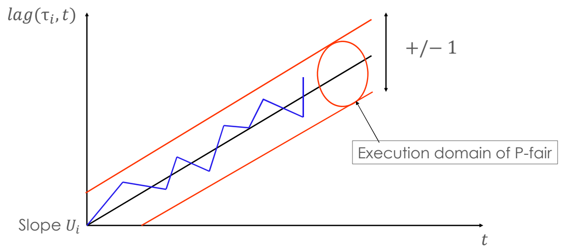


$lag(\tau_1, 1) = 1 * \left(\frac{1}{4}\right) - 1 = -\frac{3}{4}$
$lag(\tau_4, 3) = 3 * \left(\frac{1}{4}\right) - 0 = \frac{3}{4}$

$-1 < lag(\tau_i, t) < 1$ seems to be the worst-case lag

Proportionate fairness

- Definition (P-fair schedule):
  a schedule is P-fair if and only if $\forall\, \tau_i$ and $\forall\, t$: $-1 < lag(\tau_i, t) < 1$



Proportionate fairness

- Theorem
  A P-fair schedule is optimal in the sense of feasibility for a set of periodic tasks with implicit deadlines
- Proof
  A schedule S is P-fair
  $\Rightarrow -1 < lag(\tau_i, t) < 1$
  $\Rightarrow -1 < lag(\tau_i, kT_i) < 1$
  $\Rightarrow -1 < kT_i \frac{C_i}{T_i} - allocated(\tau_i, kT_i) < 1$
  $\Rightarrow -1 < kC_i - allocated(\tau_i, kT_i) < 1$
  $\Rightarrow kC_i - allocated(\tau_i, kT_i) = 0$
  $\Rightarrow kC_i = allocated(\tau_i, kT_i)$
  $\Rightarrow allocated(\tau_i, (k+1)T_i) - allocated(\tau_i, kT_i) = C_i$
  $\Rightarrow \tau_i$ executes C_i time-units during $[kT_i,\ (k+1)T_i]$
  $\Rightarrow \tau_i$ meets every deadline in periodic scheduling

The algorithm PF

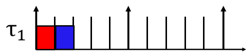
- **How to generate a P-fair schedule?**
  - Execute all *urgent* tasks
    - A task τ_i is urgent at time t if
      $lag(\tau_i, t) > 0$ and $lag(\tau_i, t+1) \geq 0$ if τ_i executes
  - Do not execute *tnegru* tasks
    - A task τ_i is tnegru at time t if
      $lag(\tau_i, t) < 0$ and $lag(\tau_i, t+1) \leq 0$ if τ_i does not execute
  - For the other tasks, execute the task that has the least t such that $lag(\tau_i, t) > 0$

The algorithm PF

- **Results**
  - The algorithm PF assigns priorities to tasks at every time slot → Job-level dynamic priority (JLDP) scheduling policy
  - Theorem: the schedule generated by algorithm PF is P-fair.
    - Proof: [Baruah et al., '96]

The algorithm PF

- Example: $\tau = \{(T_1 = 5, C_1 = 2),\ (T_2 = 5,\ C_2 = 3)\}$, one processor



At time 0, any of the two tasks may be scheduled

At time 1:

$$lag(\tau_1, 1) = 1 * \left(\frac{2}{5}\right) - 1 = -\frac{3}{5}$$

$$lag(\tau_2, 1) = 1 * \left(\frac{3}{5}\right) - 0 = \frac{3}{5}$$

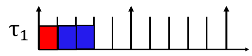
At time 2 if τ_2 executes:

$$lag(\tau_2, 2) = 2 * \left(\frac{3}{5}\right) - 1 = \frac{1}{5}$$

τ_2 is urgent at time 1!!

The algorithm PF

- Example: $\tau = \{(T_1 = 5, C_1 = 2),\ (T_2 = 5,\ C_2 = 3)\}$, one processor



At time 2:

$$lag(\tau_1, 2) = 2 * \left(\frac{2}{5}\right) - 1 = -\frac{1}{5}$$

$$lag(\tau_2, 2) = 2 * \left(\frac{3}{5}\right) - 1 = \frac{1}{5}$$

At time 3 if τ_2 executes:

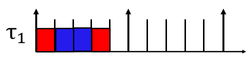
$$lag(\tau_1, 3) = 3 * \left(\frac{2}{5}\right) - 1 = \frac{1}{5}$$

$$lag(\tau_2, 3) = 3 * \left(\frac{3}{5}\right) - 2 = -\frac{1}{5}$$

τ_2 is scheduled since it has the least t such that lag is positive

The algorithm PF

- Example: $\tau = \{(T_1 = 5, C_1 = 2),\ (T_2 = 5,\ C_2 = 3)\}$, one processor



At time 3:

$$lag(\tau_1, 3) = 3 * \left(\frac{2}{5}\right) - 1 = \frac{1}{5}$$

$$lag(\tau_2, 3) = 3 * \left(\frac{3}{5}\right) - 2 = -\frac{1}{5}$$

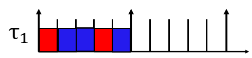
At time 4 if τ_1 executes:

$$lag(\tau_1, 4) = 4 * \left(\frac{2}{5}\right) - 2 = -\frac{2}{5}$$

τ_1 is scheduled since it has the least t such that lag is positive

The algorithm PF

- Example: $\tau = \{(T_1 = 5, C_1 = 2),\ (T_2 = 5,\ C_2 = 3)\}$, one processor



At time 4:

$$lag(\tau_1, 4) = 4 * \left(\frac{2}{5}\right) - 2 = -\frac{2}{5}$$

$$lag(\tau_2, 4) = 4 * \left(\frac{3}{5}\right) - 2 = \frac{2}{5}$$

At time 5 if τ_2 executes:

$$lag(\tau_2, 5) = 5 * \left(\frac{3}{5}\right) - 3 = 0$$

τ_2 is urgent at time 4!!

…and so on…

Proportionate fairness

- Exact test of existence of a P-fair schedule:

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

- Full processor utilization!

Disadvantages

- High number of preemptions
- High number of migrations
- Optimal only for implicit deadlines

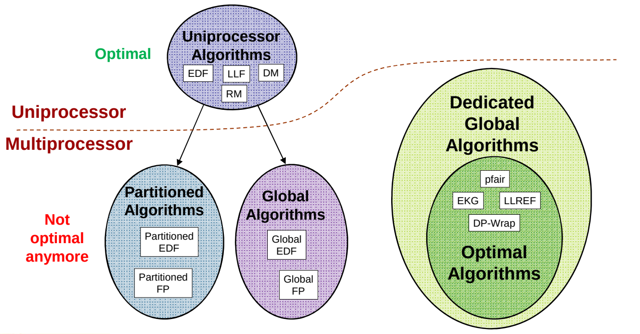
(Other) negative results

- No optimal algorithm is known for constrained or arbitrary deadline systems
- No optimal online algorithm is possible for arbitrary collections of jobs [Leung and Whitehead]
- Even for sporadic task systems, optimality requires **clairvoyance** [Fisher et al., 2009]

⇒ Many sufficient schedulability tests exist, according to different metrics of evaluation

We will see one of those in the next lecture …

Taxonomy of multiprocessor scheduling algorithms



Thank you!

Alessandra Melani
alessandra.melani@sssup.it