Global Scheduling in Multiprocessor Real-Time Systems

Alessandra Melani

Global vs Partitioned scheduling

- Single shared queue instead of multiple dedicated queues

<table>
<thead>
<tr>
<th>Global scheduling</th>
<th>Partitioned scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Global Scheduling Diagram" /></td>
<td><img src="image2" alt="Partitioned Scheduling Diagram" /></td>
</tr>
</tbody>
</table>

- 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

- **Synchronous periodic situation**

  \[
  C_i, D_i, T_i
  \]

  \[
  r_1 = (1, 1, 2) \\
  r_2 = (1, 1, 3) \\
  r_3 = (5, 6, 6)
  \]

  The second job of \( r_1 \) 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 $\tau_1$, 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 $U \times t$ 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

\[ lag(\tau_i, t) = t \times \left( \frac{C_i}{T_i} \right) - \text{allocated}(\tau_i, t) \]

Error

“Fluid” execution: should have executed in $[0, t)$

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

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- No task executes in $[0,1)$
  - $\text{lag}(\tau_1, 1) = 1 \times \frac{2}{5} - 0 \neq 0$
  - $\text{lag}(\tau_2, 1) = 1 \times \frac{3}{7} - 0 \neq 0$

- Task $\tau_1$ executes in $[0,1)$
  - $\text{lag}(\tau_1, 1) = 1 \times \frac{2}{5} - 1 \neq 0$
  - $\text{lag}(\tau_2, 1) = 1 \times \frac{3}{7} - 0 \neq 0$

- Task $\tau_2$ executes in $[0,1)$
  - $\text{lag}(\tau_1, 1) = 1 \times \frac{2}{5} - 0 \neq 0$
  - $\text{lag}(\tau_2, 1) = 1 \times \frac{3}{7} - 1 \neq 0$

$\text{lag}(\tau_i, 1)$ is impossible at time 1

Proportionate fairness

- Example: $\tau = \{T_1 = 4, C_1 = 1\}, \{T_2 = 4, C_2 = 1\}, \{T_3 = 4, C_3 = 1\}, \{T_4 = 4, C_4 = 1\}$, one processor

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
<th>$\tau_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="Diagram" /></td>
<td><img src="image4.png" alt="Diagram" /></td>
<td><img src="image5.png" alt="Diagram" /></td>
<td><img src="image6.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- $\text{lag}(\tau_1, 1) = 1 \times \frac{1}{4} - 1 = -\frac{3}{4}$
- $\text{lag}(\tau_4, 3) = 3 \times \frac{1}{4} - 0 = \frac{3}{4}$

$-1 < \text{lag}(\tau_i, t) < 1$ seems to be the worst-case lag
**Definition (P-fair schedule):**
a schedule is P-fair if and only if \( \forall \tau_i \) and \( \forall t: -1 < \text{lag}(\tau_i, t) < 1 \)

**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
\[ -1 < \text{lag}(\tau_i, t) < 1 \]
\[ -1 < \text{lag}(\tau_i, kT_i) < 1 \]
\[ -1 < kT_i \frac{C_i}{T_i} - \text{allocated}(\tau_i, kT_i) < 1 \]
\[ -1 < kC_i - \text{allocated}(\tau_i, kT_i) < 1 \]
\[ kC_i = \text{allocated}(\tau_i, kT_i) \]
\[ \text{allocated}(\tau_i, (k+1)T_i) - \text{allocated}(\tau_i, kT_i) = C_i \]
\[ \tau_i \text{ executes } C_i \text{ time-units during } [kT_i, (k + 1)T_i) \]
\[ \tau_i \text{ meets every deadline in periodic scheduling} \]
The algorithm PF

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

**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:

- $\tau_1$ executes at time 1:
  
  $\text{lag}(\tau_1, 1) = 1 \times \left(\frac{2}{5}\right) - 1 = -\frac{3}{5}$

- $\tau_2$ executes at time 1:
  
  $\text{lag}(\tau_2, 1) = 1 \times \left(\frac{3}{5}\right) - 0 = \frac{3}{5}$

At time 2 if $\tau_2$ executes:

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

$\tau_2$ is urgent at time 1!!

At time 2:

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

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

At time 3 if $\tau_2$ executes:

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

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

$\tau_2$ is scheduled since it has the least $\tau$ 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:

- $\text{lag}(\tau_1, 3) = 3 \times \left(\frac{2}{5}\right) - 1 = \frac{1}{5}$
- $\text{lag}(\tau_2, 3) = 3 \times \left(\frac{3}{5}\right) - 2 = -\frac{1}{5}$

$\tau_1$ is scheduled since it has the least $\tau$ such that lag is positive.

At time 4 if $\tau_1$ executes:

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

At time 4:

- $\text{lag}(\tau_1, 4) = 4 \times \left(\frac{3}{5}\right) - 2 = \frac{2}{5}$

...and so on...

The algorithm PF

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

At time 5 if $\tau_2$ executes:

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

$\tau_2$ is urgent at time 4!!
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]

\[ \Rightarrow \] 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

Uniprocessor Algorithms
- EDF
- LLF
- DM
- RM

Partitioned Algorithms
- Partitioned EDF
- Partitioned FP

Global Algorithms
- Global EDF
- Global FP

Dedicated Global Algorithms
- pfair
- EKG
- LLREF
- DP-Wrap

Optimal Algorithms

Uniprocessor
Not optimal anymore

Multiprocessor
Optimal

Thank you!
Alessandra Melani
alessandra.melani@sssup.it