Dependable Real-Time Systems

Lecture #9

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Multiprocessor scheduling

How are tasks assigned to processors?

- Static assignment
  - The processor(s) used for executing a task are determined before system is put in mission (“off-line”)
  - Approaches: partitioned scheduling, guided search, non-guided search, ...
- Dynamic assignment
  - The processor(s) used for executing a task are determined during system operation “on-line”
  - Approach: global scheduling

Multiprocessor scheduling

How are tasks allowed to migrate?

- Partitioned scheduling (no migration!)
  - Each instance of a task must execute on the same processor
  - Equivalent to multiple uniprocessor systems!
- Guided search & non-guided techniques
  - Depending on migration constraints, a task may or may not execute on more than one processor
- Global scheduling (full migration!)
  - A task is allowed to execute on an arbitrary processor (sometimes even after being preempted)

Global scheduling

Implementation approaches:

- Greedy scheduling
  - As used by the traditional (RM, DM, EDF) priority scheduler.
- Fair scheduling
  - As used by the p-fair (proportionate fairness) scheduler.

Performance issues:

- Weak theoretical framework
  - Key properties from uniprocessor scheduling no longer valid
- Suffers from several scheduling anomalies
  - Counter-intuitive sensitivity to change in system parameters

Global scheduling

A fundamental limit: (Andersson, Baruah & Jonsson, 2001)

> The utilization guarantee bound for multiprocessor scheduling (strictly partitioned or strictly global), using task priorities only, cannot be higher than 50% of the processing capacity.

- One (older) approach to circumvent this limit is to use p-fair (priorities + time quanta) scheduling and dynamic task priorities.
- Another (newer) approach to circumvent this limit is to split some of the tasks into two or more subtasks, and then run each subtask of a split task on a separate processor. The remaining tasks use partitioned scheduling.

Global scheduling

Complexity of schedulability analysis for global scheduling: (Leung & Whitehead, 1982)

The problem of deciding whether a task set (synchronous or asynchronous) is schedulable on m processors with respect to global scheduling is NP-complete in the strong sense.

Consequence:

There can only exist a pseudo-polynomial time algorithm for

(i) finding an optimal static priority assignment, or

(ii) exact feasibility testing

But not both at the same time!

Weak theoretical framework

Underlying causes:

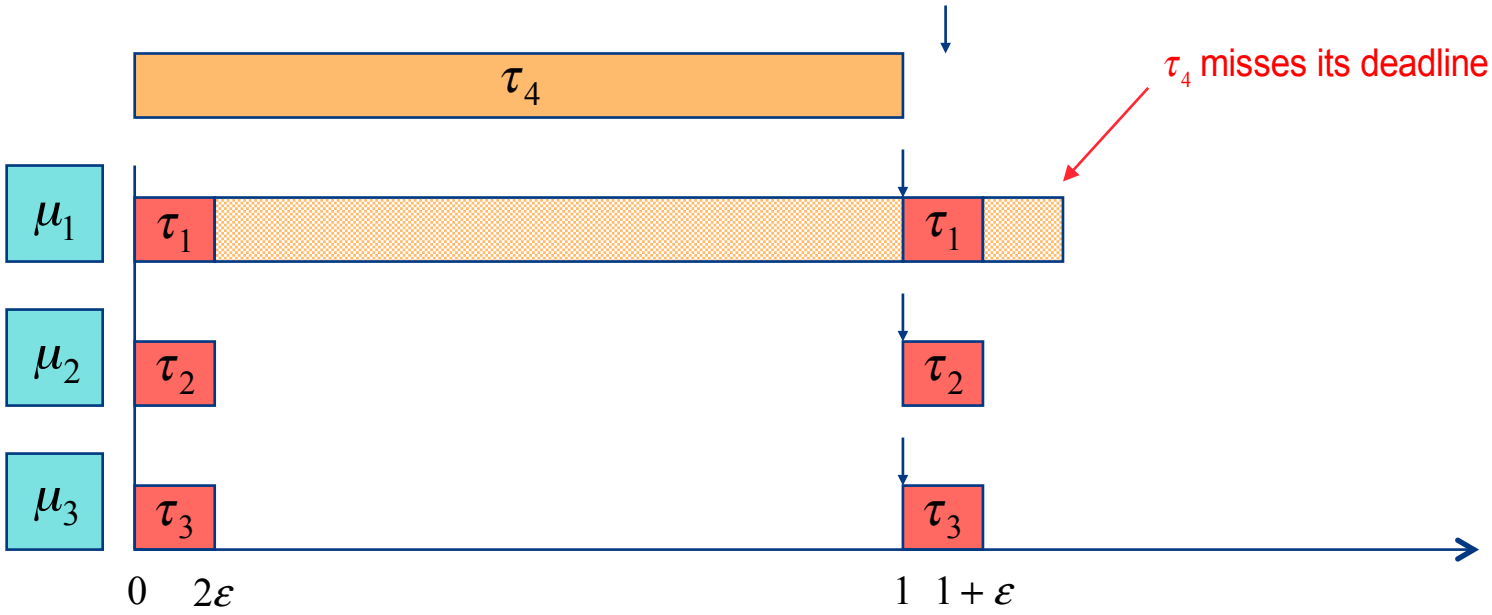
- Dhall's effect:
  - With greedy scheduling, a low-utilization task set may be unschedulable regardless of how many processors are used.
- Hard-to-find critical instant:
  - A critical instant does not always occur when a task arrives at the same time as all its higher-priority tasks.
- Dependence on relative priority ordering:
  - Changing the relative priority ordering among higher-priority tasks may affect schedulability for a lower-priority task.

Weak theoretical framework

Dhall's effect: (Dhall & Liu, 1978)

$$\tau_1 = \{C_1 = 2\varepsilon, T_1 = 1\}$$
$$\tau_2 = \{C_2 = 2\varepsilon, T_2 = 1\}$$
$$\tau_3 = \{C_3 = 2\varepsilon, T_3 = 1\}$$
$$\tau_4 = \{C_4 = 1, T_4 = 1+\varepsilon\}$$

RM scheduling



Weak theoretical framework

Dhall's effect:

- Applies for (greedy) RM, DM and EDF scheduling
- Least utilization of unschedulable task sets can be arbitrarily close to 1 no matter how many processors are used.

$$U_{global} = m\frac{2\varepsilon}{1} + \frac{1}{1+\varepsilon} \rightarrow 1$$

when $\varepsilon \rightarrow 0$

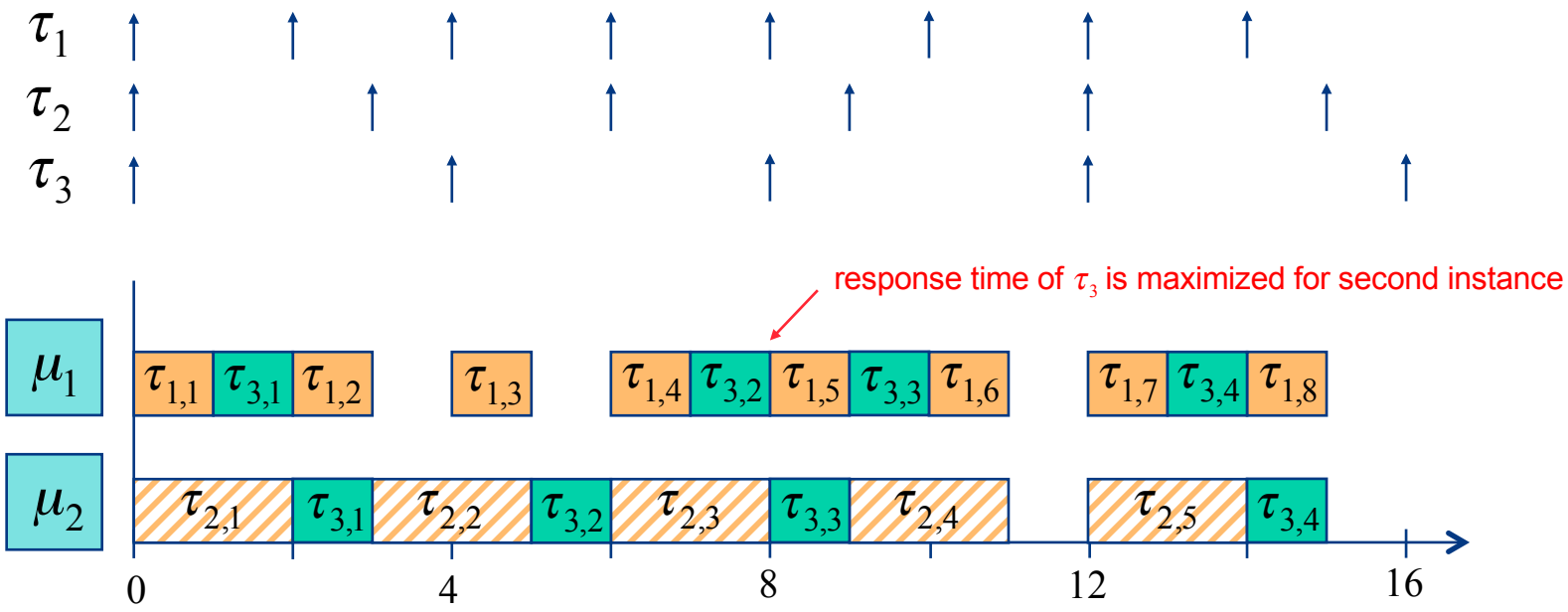
Conclusion in year 2000: new priority-assignment policies are needed for effective resource utilization of multiprocessors!

Weak theoretical framework

Hard-to-find critical instant:

$$\tau_1 = \{C_1 = 1, T_1 = 2\}$$
$$\tau_2 = \{C_2 = 2, T_2 = 3\}$$
$$\tau_3 = \{C_3 = 2, T_3 = 4\}$$

RM scheduling



Weak theoretical framework

Hard-to-find critical instant: (Andersson & Jonsson, 2000)

- A critical instant does not always occur when a task arrives at the same time as all its higher-priority tasks.
- Recall that an easy-to-find critical instant enabled the design of many efficient uniprocessor feasibility tests.
- A hard-to-find critical instant makes it challenging to design efficient multiprocessor feasibility tests, as it will
  - incur a high time complexity (to find critical instant), or
  - suffer from pessimism (due to interference uncertainty)

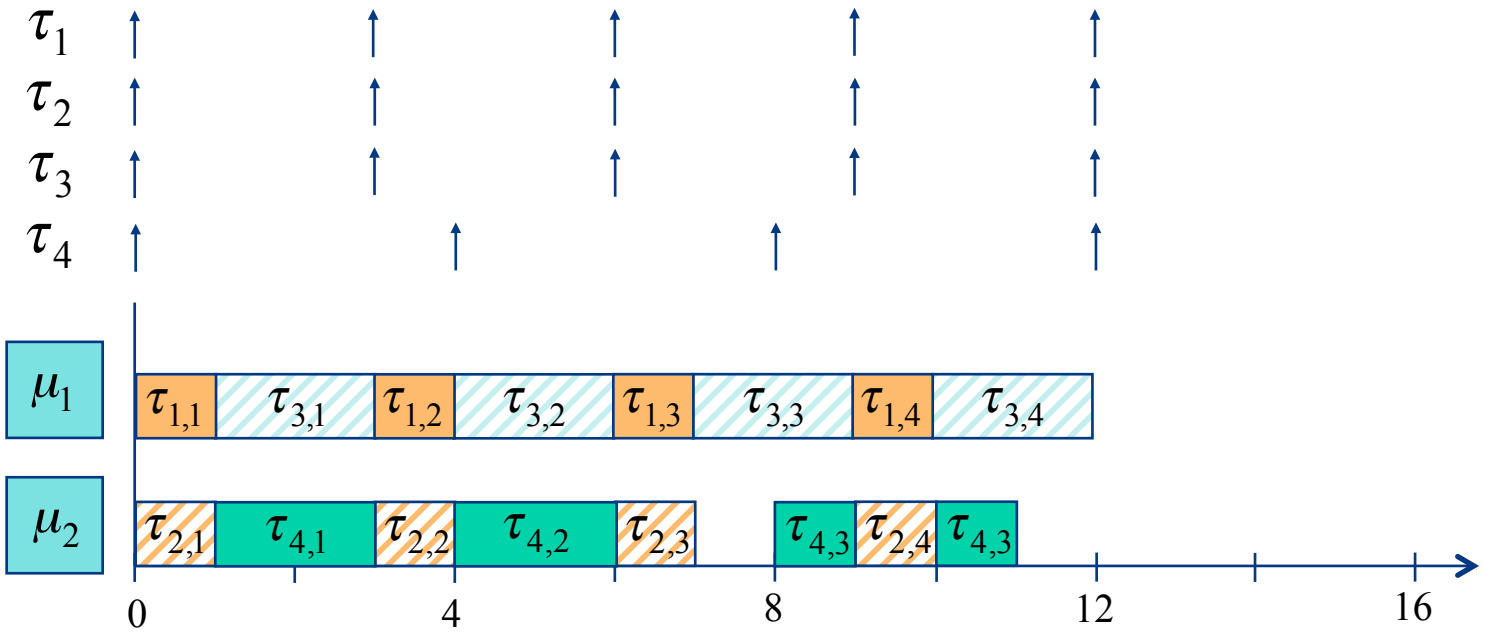
Conclusion in year 2000: new insights are needed for designing efficient feasibility tests for multiprocessors!

Weak theoretical framework

Impact of relative priority ordering:

$$\tau_1 = \{C_1 = 1, T_1 = 3\}$$
$$\tau_2 = \{C_2 = 1, T_2 = 3\}$$
$$\tau_3 = \{C_3 = 2, T_3 = 3\}$$
$$\tau_4 = \{C_1 = 2, T_1 = 4\}$$

RM scheduling
(priority order follows task index)

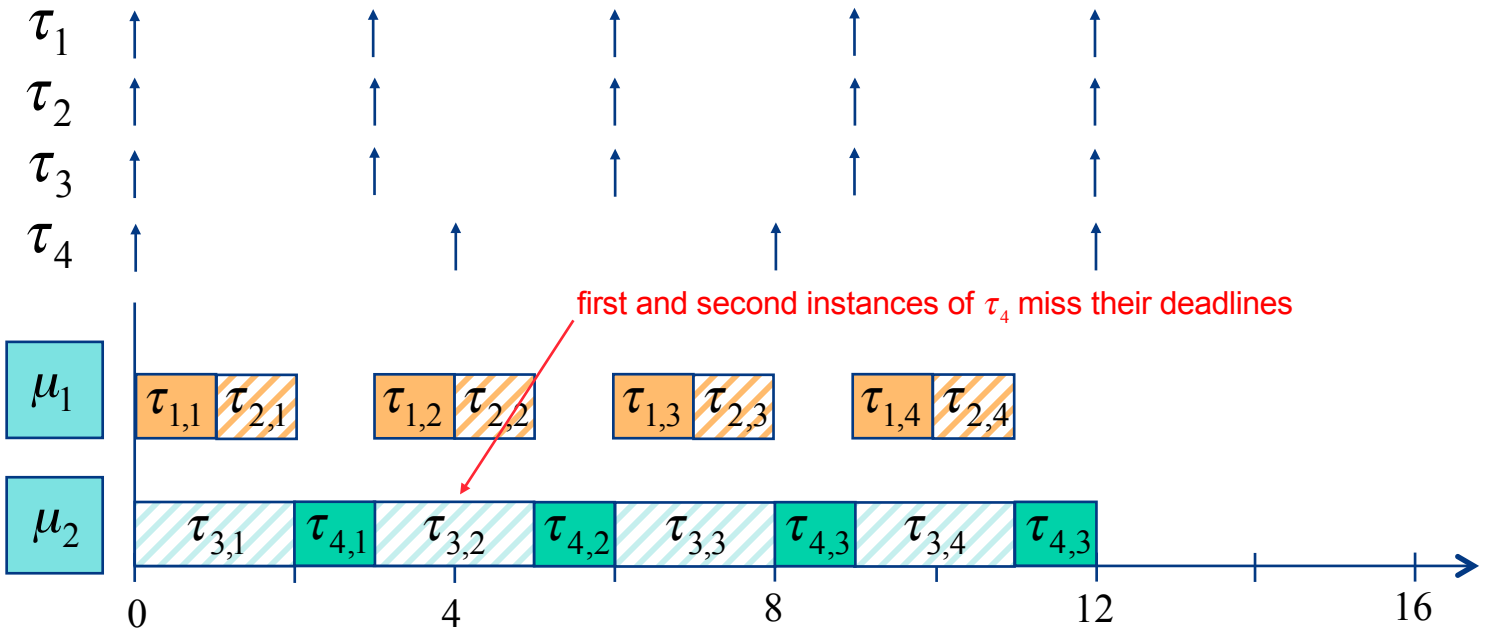


Weak theoretical framework

Impact of relative priority ordering:

$$\tau_1 = \{C_1 = 1, T_1 = 3\}$$
$$\tau_2 = \{C_2 = 1, T_2 = 3\}$$
$$\tau_3 = \{C_3 = 2, T_3 = 3\}$$
$$\tau_4 = \{C_1 = 2, T_1 = 4\}$$

RM scheduling
(priorities of τ_2 and τ_3 swapped)



Weak theoretical framework

Impact of relative priority ordering: (Andersson & Jonsson, 2000)

- The response time of a task may depend on the relative priority ordering of the higher-priority tasks.
- This property does not exist for a uniprocessor system.
- This means that the OPA algorithm, which can be used on a uniprocessor for finding an optimal priority assignment, may not have that capability on a multiprocessor system.

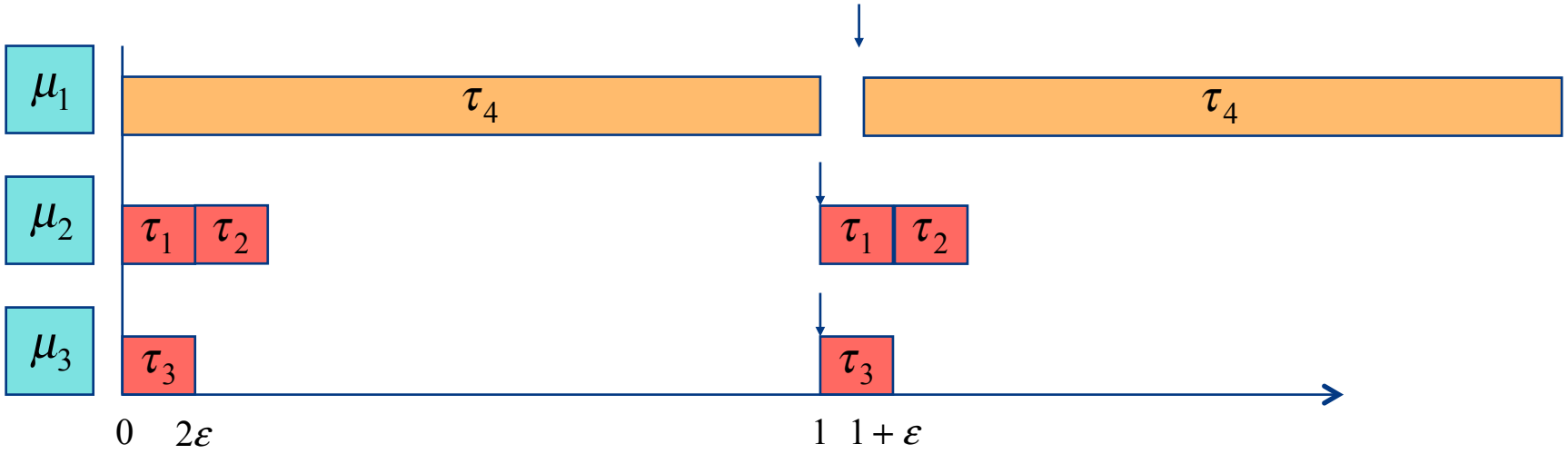
Conclusion in year 2000: new insights are needed of how to best use the OPA algorithm for deriving optimal priority assignments for multiprocessors!

Improved resource utilization

How to avoid Dhall's effect:

Insight #1: RM, DM and EDF only account for task deadlines!
Actual computation demands are not accounted for.

Insight #2: Dhall's effect can be avoided in greedy scheduling by letting tasks with high utilization receive higher priority:



Improved resource utilization

Scientific breakthrough: (Andersson, Baruah & Jonsson, 2001)

RM-US{m/(3m-2)}

- RM-US{m/(3m-2)} assigns (static) priorities to tasks according to the following rule:

  If $U_i > m/(3m-2)$ then τ_i has the highest priority (ties broken arbitrarily)

  If $U_i \leq m/(3m-2)$ then τ_i has RM priority

- Clearly, tasks with higher utilization get higher priority.

Improved resource utilization

Scientific breakthrough:

Guarantee bound analysis for RM-US{m/(3m-2)}:

- A sufficient condition for RM-US{m/(3m-2)} scheduling on m identical processors is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \frac{m^2}{3m-2}$$

- Question: does RM-US{m/(3m-2)} avoid Dhall's effect?

Improved resource utilization

Scientific breakthrough:

Guarantee bound analysis for RM-US{m/(3m-2)}:

- We observe that, regardless of the number of processors, the task set will always meet its deadlines as long as no more than one third of the processing capacity is used:

$$U_{RM-US\{m/(3m-2)\}} = \lim_{m\to\infty} \frac{m^2}{3m-2} = \frac{m}{3}$$

- RM-US{m/(3m-2)} thus avoids Dhall's effect since we can always add more processors if deadlines were missed.

Improved resource utilization

State-of-the-art guarantee bounds for global scheduling:

- Static priorities: ("greedy" global scheduling)
  - The SM-US{2/(3+√5)} priority-assignment policy has a guarantee bound of 38.2%. (Andersson, 2008)
- Dynamic priorities: ("greedy" global scheduling)
  - The EDF-US{m/(2m-1)} priority-assignment policy has a guarantee bound of 50%. (Srinivasan & Baruah, 2002)
- Task splitting: ("greedy" global+partitioned scheduling)
  - The SPA2 task-splitting algorithm has a guarantee bound of 69.3% (c.f. the RM bound for uniprocessors). (Guan, et al., 2010)
- Optimal multiprocessor scheduling: ("fair" global scheduling)
  - P-fair scheduling using dynamic priorities has a guarantee bound of 100%. (Baruah et al., 1996)

Sufficient response-time test

Scientific breakthrough: (Andersson & Jonsson, 2000)

Response-time analysis for global scheduling:

- The response time for task τ_i is (as before): $R_i = C_i + I_i$

  C_i : the task WCET

  I_i : the interference from higher-priority tasks

$$I_i = \frac{1}{m} \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + C_j \right)$$

where $hp(i)$ is the set of tasks with higher priority than τ_i

Sufficient response-time test

Scientific breakthrough:

Response-time analysis for global scheduling:

- As before, an iterative approach can be used for finding the worst-case response time:

$$R_i^{n+1} = C_i + \frac{1}{m} \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^n}{T_j} \right\rceil \cdot C_j + C_j \right)$$

- The sufficient feasibility test is then:

$$\forall i : R_i \leq D_i$$

Sufficient response-time test

Scientific breakthrough:

Response-time analysis for global scheduling:

- The extra execution-time term introduced in this analysis is an example of carry-in interference.

$$I_i = \frac{1}{m} \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j + C_j \right)$$

- Significant research efforts have been made in recent years to derive accurate estimates of the worst-case carry-in interference, and utilize these estimates in improved feasibility tests for global scheduling.

Optimal priority assignment

Scientific breakthrough:

Conditions for OPA compatibility: (Davis & Burns, 2009)

Condition 1: The schedulability of a task τ may, according to test S, depend on any independent properties of tasks with priorities higher than τ, but not on any properties of those tasks that depend on their relative priority ordering.

Condition 2: The schedulability of a task τ may, according to test S, depend on any independent properties of tasks with priorities lower than τ, but not on any properties of those tasks that depend on their relative priority ordering.

Condition 3: When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test S, if it was previously schedulable at the lower priority.

Optimal priority assignment

Scientific breakthrough:

Conditions for OPA compatibility:

- Task properties are referred to as independent if they have no dependency on the priority assigned to the task.
  (e.g. WCET, period, deadline)
- Task properties are referred to as dependent if they have a dependency on the priority assigned to the task.
  (e.g. worst-case response time)
- Feasibility tests which satisfy these conditions can be used together with the OPA algorithm to derive an optimal priority assignment for global static-priority scheduling.

Optimal priority assignment

Scientific breakthrough:

State-of-the-art in OPA compatibility:

- Any exact test for global static-priority preemptive scheduling of strictly periodic task systems is incompatible with OPA.
  (Davis and Burns, 2011)
- Any exact test for global static-priority preemptive scheduling of sporadic task systems is incompatible with OPA.
  (Davis and Burns, 2016)

For these exact tests, the only currently-known optimal priority assignment policy is to check all *n!* possible priority orderings.

Optimal priority assignment

Scientific breakthrough:

State-of-the-art in OPA compatibility:

- The sufficient response-time analysis presented in this lecture is OPA-compatible for global static-priority scheduling.
- A few other feasibility tests are known to be OPA-compatible for global static-priority scheduling. Common for these tests is that they use a novel strategy for calculating interference.

Highlighted article:

Read the paper by Davis and Burns (RTS Journal, 2011)

Study particularly how the strategy with a ‘problem window’ is used for deriving a sufficient schedulability test (Section 3)

P-fair scheduling: an overview

Underlying principles:

- Proportionate progress:
  - The guiding rule for a p-fair scheduler is to execute tasks based on the concept of proportionate progress.
  - Ideally: a task with utilization C_i / T_i would be scheduled to execute a fraction C_i / T_i of a time unit per time unit.
- Quantum based:
  - Any scheduler implementation manages the execution of tasks in atomic time units, meaning that a task must execute during a whole time unit or not execute during that time unit at all.
  - Challenge: how to approximate ideal proportionate progress if a task switch can only take place at time unit boundaries?

P-fair scheduling: an overview

Approximated proportionate progress:

- The concept of lag:
  - For each time unit t compare how much a task τ_i should have (ideally) executed during $[0,t)$ with how much the task actually have executed during $[0,t)$:

$$\text{lag}(\tau_i,t) = t \cdot (C_i / T_i) - \text{allocated}(\tau_i,t)$$

- Consequences:
  - If τ_i executes, then $\text{lag}(\tau_i,t)$ decreases by $1 - C_i / T_i$
  - If τ_i does not execute, then $\text{lag}(\tau_i,t)$ increases by C_i / T_i
  - Note: It is not always possible to have $\text{lag}(\tau_i,t) = 0$

P-fair scheduling: an overview

P-fair scheduling:

- Definition:
  - A schedule is p-fair if, and only if:

$$\forall i,t : -1 < \mathrm{lag}(\tau_i, t) < 1$$

- Consequences:
  - If a schedule is p-fair then it is also a feasible solution to the periodic task scheduling problem!
  - Note: In a p-fair schedule the periodic execution of tasks will have low jitter (= small deviations from expected periodicity)

P-fair scheduling: an overview

P-fair scheduling:

- Existence of a p-fair schedule:
  - For a system with n periodic tasks and m identical processors:

$$\sum_{i=1}^{n} C_i / T_i \leq m \quad \Rightarrow \text{ a p-fair schedule exists}$$

- Assumptions:
  - Fully preemptive scheduling (no preemption cost)
  - Full task migration (no migration cost)
  - Synchronous task set
  - Independent tasks
  - Periodic tasks, where $D_i = T_i$ for all tasks

P-fair scheduling: an overview

Algorithm PF: (Baruah et al., 1996)

- Properties:
  - Generates a p-fair schedule, by using task priorities that take into account the lag of a task.
  - Task priorities are fully dynamic, in the sense that the priority order of the tasks may change from one time unit to the next
- Scheduling approach:

  For each time unit:
  - Identify the urgent and non-urgent tasks
  - Execute all urgent tasks (n_u tasks)
  - Do not execute any of the non-urgent tasks
  - For all other tasks: execute the $m - n_u$ highest-priority ones