CHALMERS
UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG

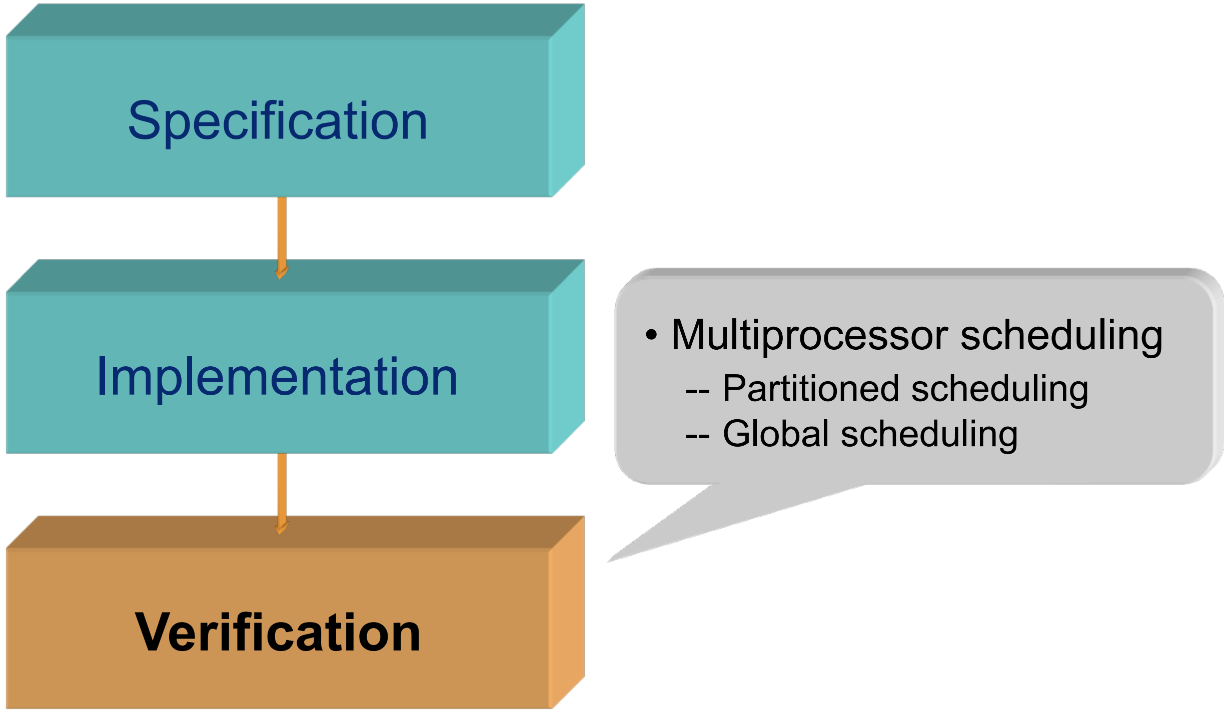
Real-Time Systems

Lecture #14

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Real-Time Systems



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")
  - Approach: Partitioned scheduling
- 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!
- Global scheduling
  - Full migration!
  - A task is allowed to execute on an arbitrary processor
  - Migration can occur even during execution of an instance of a task (for example, after being preempted)

Multiprocessor scheduling

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

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

- Hence, we should not expect to utilize more than half the processing capacity if hard real-time constraints exist.
- A way to circumvent this limit is to use p-fair (priorities + time quanta) scheduling and dynamic task priorities.

Partitioned scheduling

General characteristics:

- Each processor has its own queue for ready tasks
- Tasks are organized in groups, and each task group is assigned to a specific processor
  - For example, using a bin-packing algorithm
- When selected for execution, a task can only be dispatched to its assigned processor

Partitioned scheduling

Advantages:

- Mature scheduling framework
  - Most uniprocessor scheduling theory also applicable here
  - Uniprocessor resource-management protocols can be used
- Supported by automotive industry
  - AUTOSAR prescribes partitioned scheduling

Disadvantages:

- Cannot exploit all unused execution time
  - Surplus capacity cannot be shared among processors
  - Will suffer from overly-pessimistic WCET derivation

Partitioned scheduling

Bin-packing algorithms:

- Basic idea:
  - The problem concerns packing objects of varying sizes in boxes ("bins") with the objective of minimizing number of used boxes.
- Application to multiprocessor systems:
  - Bins are represented by processors and objects by tasks.
  - The decision whether a processor is "full" or not is derived from a utilization-based feasibility test.
- Assumptions:
  - Independent, periodic tasks
  - Preemptive, uniprocessor scheduling (RM)

Partitioned scheduling

Bin-packing algorithms:

Rate-Monotonic-First-Fit (RMFF): (Dhall and Liu, 1978)

- Let the processors be indexed as $\mu_1, \mu_2, \ldots, \mu_m$
- Assign tasks in order of increasing periods (i.e., RM order).
- For each task τ_i, choose the lowest previously-used j such that τ_i, together with all tasks that have already been assigned to processor μ_j, can be feasibly scheduled according to the utilization-based RM-feasibility test.

If all tasks are successfully assigned using RMFF, then the tasks are schedulable on *m* processors.

Partitioned scheduling

Processor utilization analysis for RMFF:

- A sufficient condition for partitioned RMFF scheduling of synchronous task sets with n tasks on m processors is

$$\boxed{U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq m\left(2^{1/2} - 1\right)}$$

(Oh & Baker, 1998)

Note: $U_{RMFF} = m\left(2^{1/2} - 1\right) \approx 0.41m$

Thus: task sets whose utilization do not exceed ≈ 41% of the total processor capacity is always RMFF-schedulable.

Partitioned scheduling

Processor utilization analysis for RMFF:

1. All tasks are independent.
2. All tasks are periodic or sporadic.
3. All tasks have identical offsets.
4. Task deadline equals the period ($D_i = T_i$).
5. Task preemptions are allowed.
6. All processors are identical.
7. Task migrations are not allowed.

Global scheduling

General characteristics:

- All ready tasks are kept in a common (global) queue that is shared among the processors
- Whenever a processor becomes idle, a task from the global queue is selected for execution on that processor.
- After being preempted, a task may be dispatched to a processor other than the one that started executing the task.

Global scheduling

Advantages:

- Supported by most multiprocessor operating systems
  - Windows 10, MacOS X, Linux, ...
- Effective utilization of processing resources
  - Unused processor time can easily be reclaimed, for example when a task does not execute its full WCET.

Disadvantages:

- Weak theoretical framework
  - Few results from the uniprocessor analysis can be used

Weak theoretical framework

The ”root of all evil” in global scheduling: (Liu, 1969)

Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that *a task can use only one processor even when several processors are free at the same time* adds a surprising amount of difficulty to the scheduling of multiple processors.

Weak theoretical framework

Underlying causes:

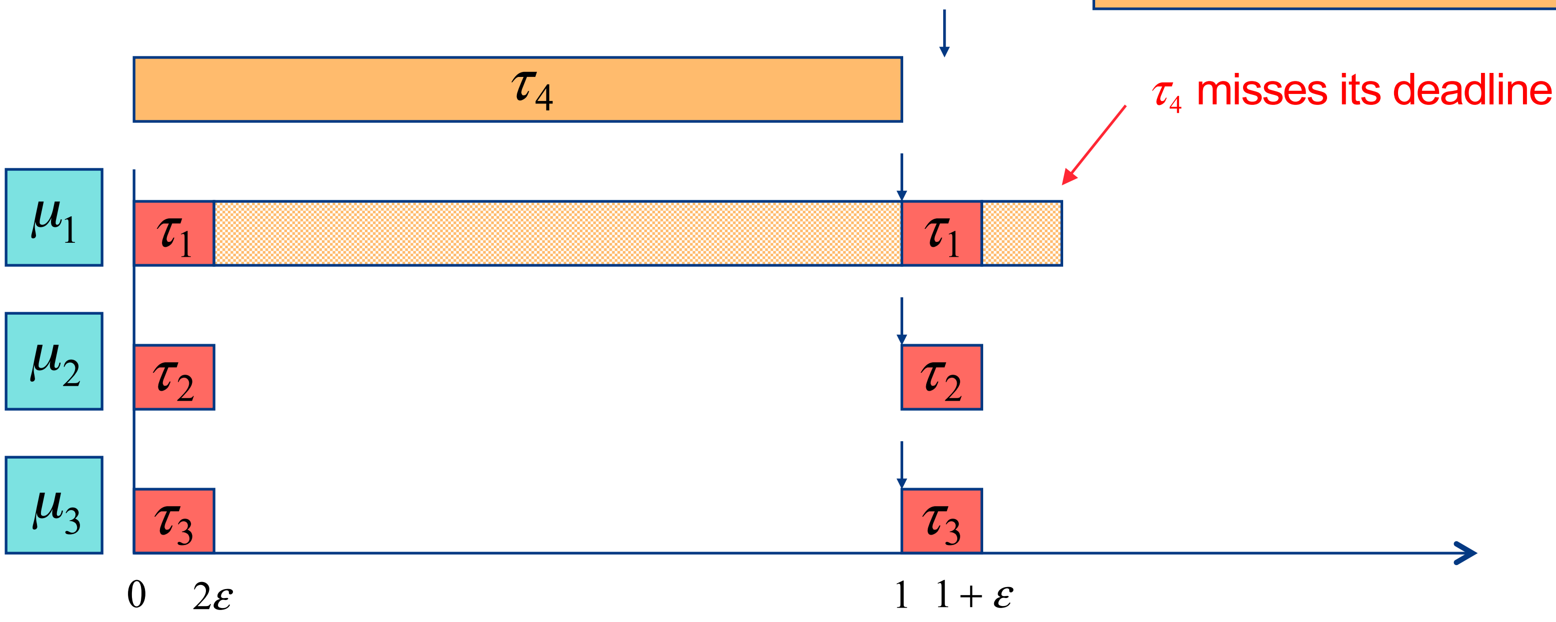
- Dhall's effect:
  - With RM, DM and EDF, some low-utilization task sets can be non-schedulable regardless of how many processors are used. Thus, any utilization guarantee bound would become so low that it would be useless in practice.
  - This is in contrast to the uniprocessor case, where we have utilization guarantee bounds of 69.3% (RM) and 100% (EDF).
- 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.
  - This is in contrast to the uniprocessor case with RM and DM (and any other static-priority policy).

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 RM, DM and EDF scheduling
- The utilization of a non-schedulable task set can be as low as to 1 (= 100%) no matter how many processors are used.

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

when $\varepsilon \rightarrow 0$

Note: Total available processor capacity is m (= $m \cdot 100\%$)

Consequence:

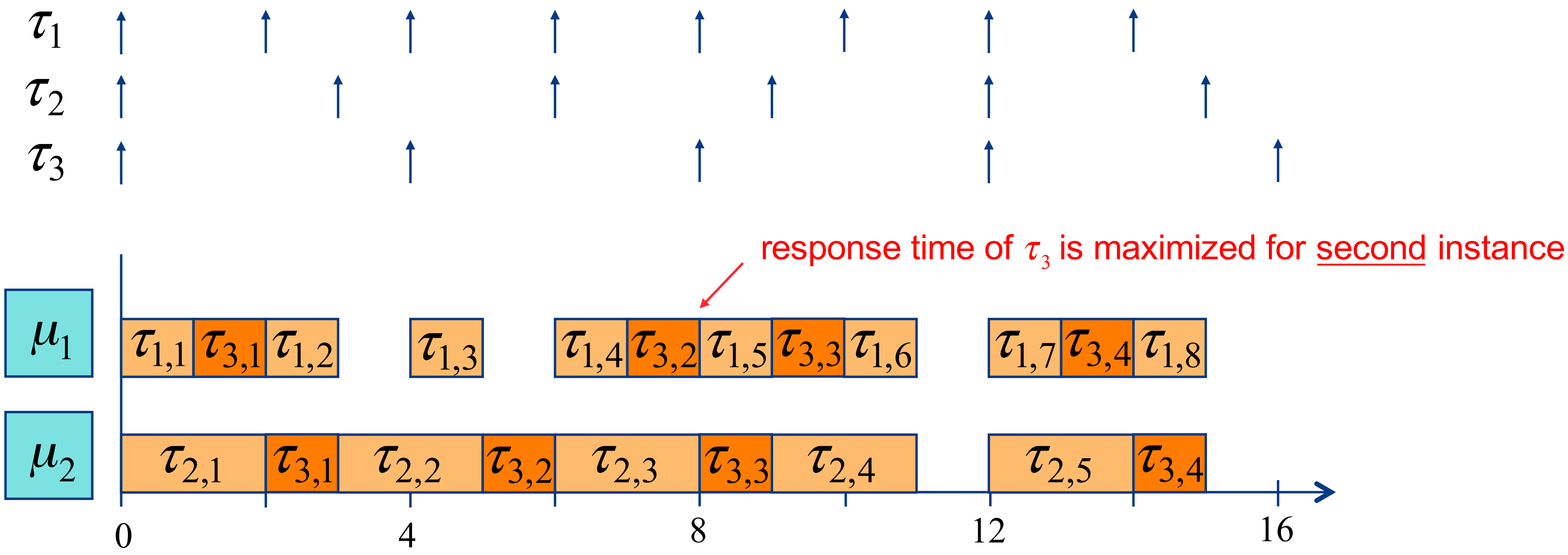
New multiprocessor priority-assignment schemes are needed!

Weak theoretical framework

Hard-to-find critical instant:

(RM scheduling)

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



Weak theoretical framework

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.
- Finding the critical instant is, in general, a problem with exponential time complexity
- Note: recall that knowledge about an easy-to-find critical instant is a fundamental assumption in the uniprocessor feasibility tests for static priority scheduling.

Consequence:

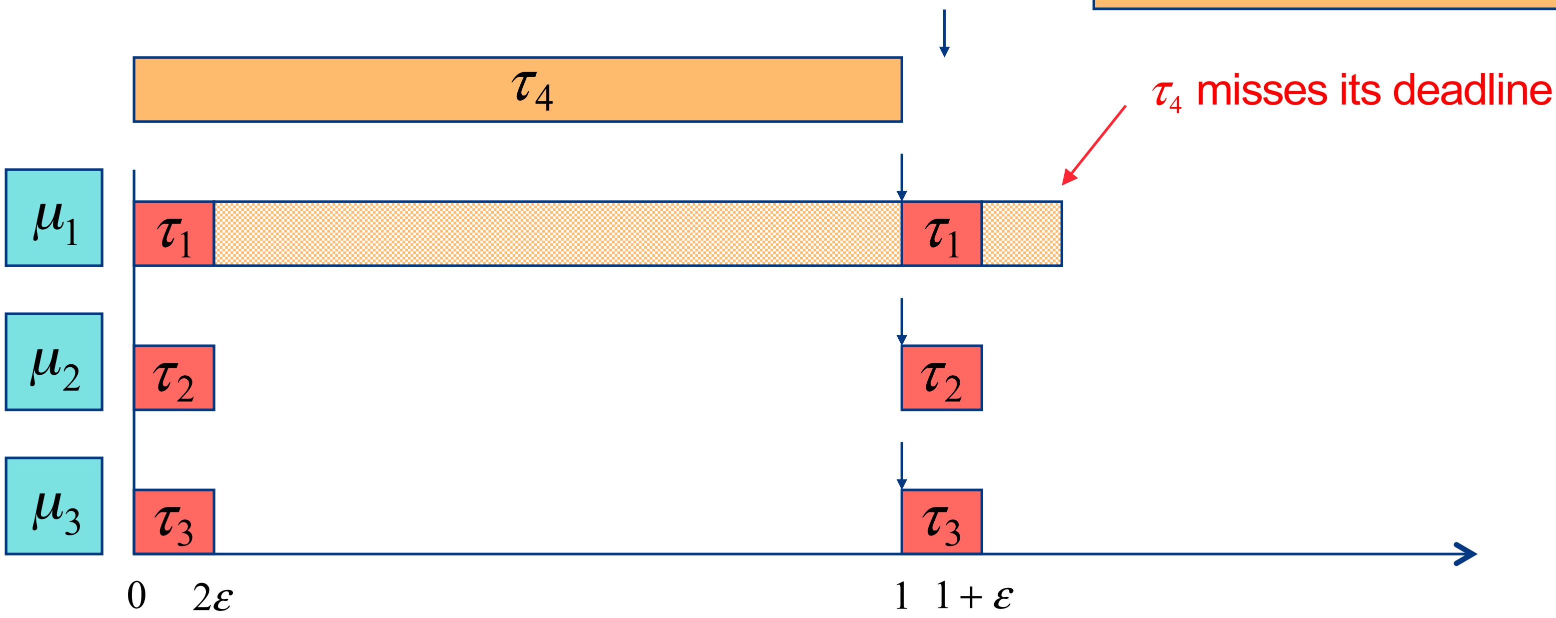
New methods for constructing effective multiprocessor feasibility tests are needed!

Weak theoretical framework

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

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

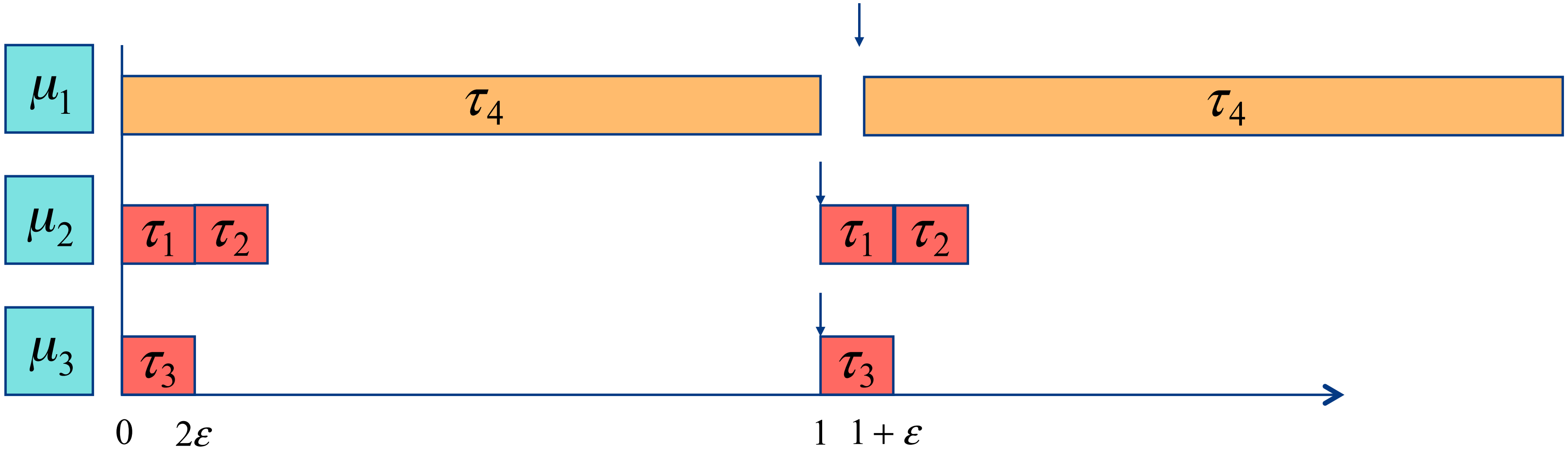
(RM scheduling)



New priority-assignment scheme

How to avoid Dhall's effect:

- Problem: RM, DM & EDF only account for task deadlines! Actual computation demands are not accounted for.
- Solution: Dhall's effect can easily be avoided by letting tasks with high utilization receive higher priority:



New priority-assignment scheme

RM-US[m/(3m-2)]: (Andersson, Baruah & Jonsson, 2001)

- 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 $U_i = C_i / T_i$ get higher priority.

Example: RM-US[m/(3m-2)]

RM-US[m/(3m-2)] example:

Assign priorities according to RM-US[m/(3m-2)], assuming the following task set to be scheduled on 3 processors:

$$\tau_1 : \{ C_1 = 1, T_1 = 7 \} \qquad \tau_2 : \{ C_2 = 2, T_2 = 10 \}$$

$$\tau_3 : \{ C_3 = 9, T_3 = 20 \} \qquad \tau_4 : \{ C_4 = 11, T_4 = 22 \}$$

$$\tau_5 : \{ C_5 = 2, T_5 = 25 \}$$

Example: RM-US[m/(3m-2)]

RM-US[m/(3m-2)] example:

- The utilizations of these tasks are: 0.143, 0.2, 0.45, 0.5 and 0.08, respectively.

For $m = 3$:

$$m/(3m-2) = 3/7 \approx 0.4286$$

- Hence, tasks τ_3 and τ_4 will be assigned higher priorities, and the remaining tasks will be assigned RM priorities.

- The possible priority assignments are therefore as follows (highest-priority task listed first):

$$\tau_3, \tau_4, \tau_1, \tau_2, \tau_5 \quad \text{or} \quad \tau_4, \tau_3, \tau_1, \tau_2, \tau_5$$

New feasibility tests

Processor utilization analysis for RM-US[m/(3m-2)]:

- A sufficient condition for RM-US[m/(3m-2)] scheduling of synchronous task sets with n tasks on m processors is

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

(Andersson, Baruah & Jonsson, 2001)

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

New feasibility tests

Processor utilization 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.
- Note that this remedy was not possible with traditional RM.

New feasibility tests

Processor utilization analysis for RM-US[m/(3m-2)]:

1. All tasks are independent.
2. All tasks are periodic (i.e. not applicable for sporadic tasks)
3. All tasks have identical offsets.
4. Task deadline equals the period ($D_i = T_i$).
5. Task preemptions are allowed.
6. All processors are identical.
7. Task migrations are allowed.

New feasibility tests

Response-time analysis for multiprocessors:

- Uses the same principle as the uniprocessor case, where the response time for a task τ_i consists of:

  C_i The task's uninterrupted execution time (WCET)

  I_i Interference from higher-priority tasks

$$\boxed{R_i = C_i + I_i}$$

- The difference is that the calculation of interference now has to account for the fact that higher-priority tasks can execute in parallel on the processors.

New feasibility tests

Response-time analysis for multiprocessors:

- For the multiprocessor case, with n tasks and m processors, we observe two things:

  1. Interference can only occur when $n > m$.
  2. Interference can only affect the $n - m$ tasks with lowest priority since the m highest-priority tasks will always execute in parallel without contention on the m processors.

- Consequently, interference of a task is a function of the <u>execution overlap</u> of its higher-priority tasks.

New feasibility tests

Response-time analysis for multiprocessors:

- The following two observations give us the secret to analyzing the interference of a task:

With respect to the execution overlap it can be shown that the interference is maximized when the higher-priority tasks completely overlap their execution.

Compared to the uniprocessor case, one extra instance of each higher-priority task must be accounted for in the interference analysis.
(due to the uncertainty regarding the critical instant).

New feasibility tests

Response-time analysis for multiprocessors:

- The worst-case interference term is

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

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

- The worst-case response time for a task τ_i is thus:

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

New feasibility tests

Response-time analysis for multiprocessors:

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

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

- We now have a sufficient condition for static-priority scheduling of periodic tasks on identical processors:

$$\boxed{\forall i : R_i \leq D_i}$$

New feasibility tests

Response-time analysis for multiprocessors:

1. All tasks are independent.
2. All tasks are periodic (i.e. not applicable for sporadic tasks)
3. All tasks have identical offsets.
4. Task deadline does not exceed the period ($D_i \leq T_i$).
5. Task preemptions are allowed.
6. All processors are identical.
7. Task migrations are allowed.

Global scheduling

Early breakthrough results in global scheduling:

- Static priorities:
  - 2001: RM-US[m/(3m-2)] circumvents Dhall's effect and has non-zero resource utilization guarantee bound of m/(3m-2) ≥ 33.3%.
  - 2003: Baker generalized the RM-US results to DM.
- Dynamic priorities:
  - 2002: Srinivasan & Baruah proposed the EDF-US[m/(2m-1)] scheme with a corresponding non-zero resource utilization guarantee bound of m/(2m-1) ≥ 50%.
- Optimal multiprocessor scheduling:
  - 1996: Baruah *et al.* proposed p-fair (priorities + time quanta) scheduling and dynamic priorities as an approach to achieve 100% resource guarantee bound on a multiprocessor.