Dependable Real-Time Systems

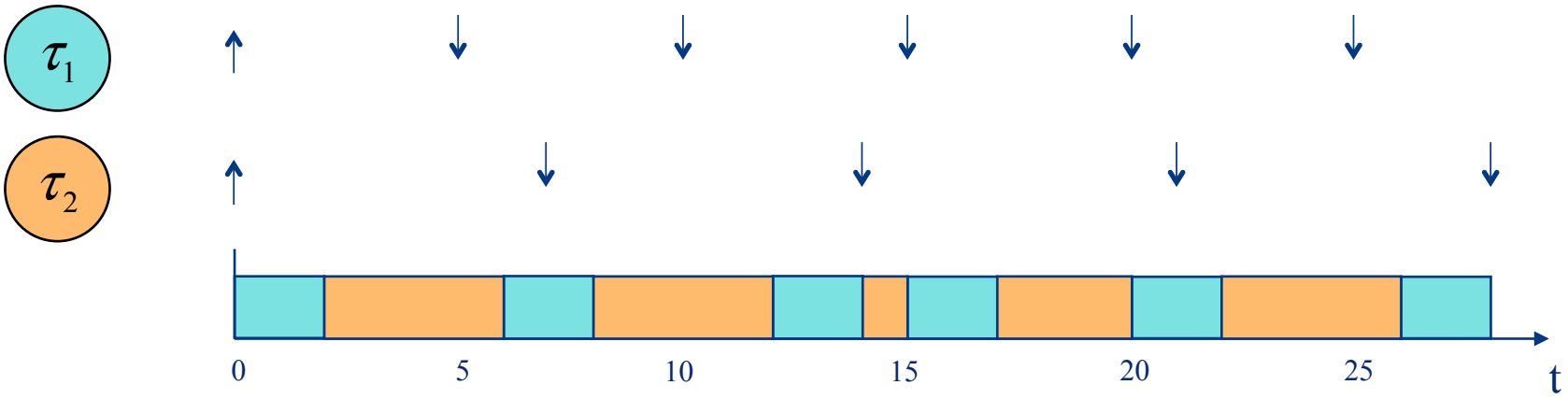
Lecture #5

Professor Jan Jonsson

Department of Computer Science and Engineering
Chalmers University of Technology

Evaluating a real-time system

An important part of real-time system design is to have techniques that generate good schedules.



Is this a good schedule? What do we need to decide the quality?

Evaluating a real-time system

Recall from an earlier lecture that:

A scheduling algorithm is said to be optimal with respect to a performance metric if it can always find a schedule that maximizes/minimizes that metric value.

What performance metrics exist for real-time systems?

Performance metrics

Traditional performance metrics:

Throughput

Average # of operations/data processed by system per time unit

Reliability

Probability that system will not fail in a given time interval

Availability

Fraction of time for which system is up (providing service)

Makespan

Length of schedule for non-periodic task graph

These metrics do not take deadlines into account!

Performance metrics

Suitable real-time performance metrics:

Laxity $X = \min_{\tau_i \in \mathbf{T}} \{ D_i - C_i \}$

Amount of time that the start of a task can be delayed without it missing its deadline (calculated before scheduling)

Lateness $L = \max_{\tau_i \in \mathbf{T}} \{ R_i - D_i \}$

Amount of time by which a task completes after its deadline (calculated after scheduling)

Successful tasks $N_{\text{success}} = \left| \{ \tau_i \in \mathbf{T} : R_i - D_i \leq 0 \} \right|$

Number of tasks that complete on or before their deadline (calculated after scheduling)

Jitter $J_{\text{output}} = \max_{\tau_i \in \mathbf{T}, k \geq 1} \left\{ \left| \left(f_{i,k+1} - f_{i,k} \right) - T_i \right| \right\}$

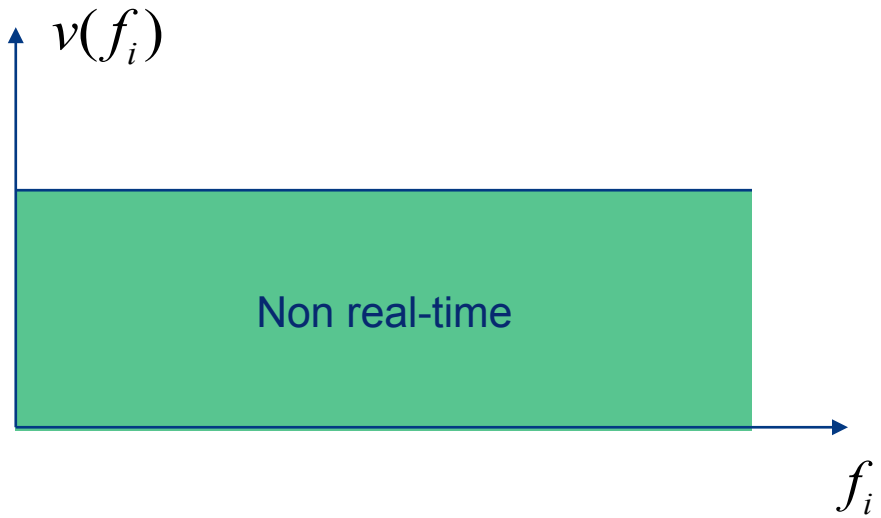
Amount of deviation from expected periodicity of a task's completion (calculated after scheduling)

Performance metrics

Cost function – a general real-time performance metric

Cumulative value: $C = \sum_{\tau_i \in \mathbf{T}} v(f_i)$

Value associated with a task as a function of its completion time

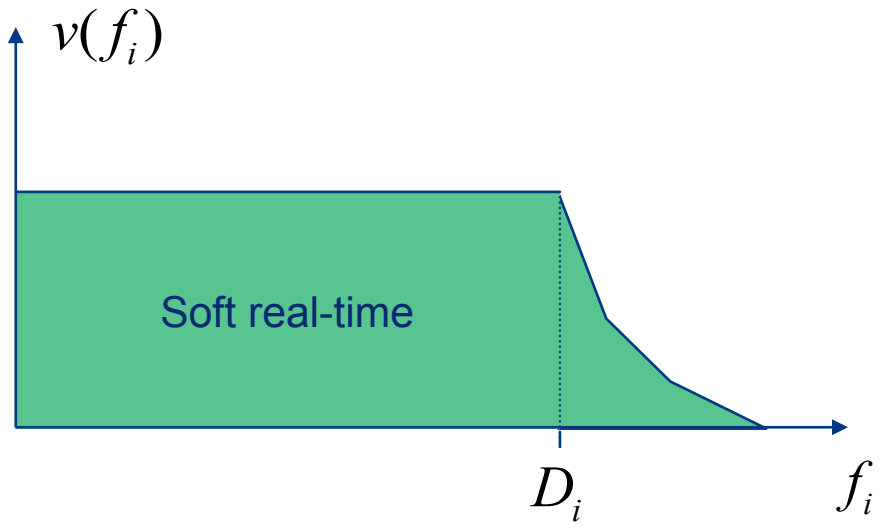


Performance metrics

Cost function – a general real-time performance metric

Cumulative value: $C = \sum_{\tau_i \in \mathbf{T}} v(f_i)$

Value associated with a task as a function of its completion time

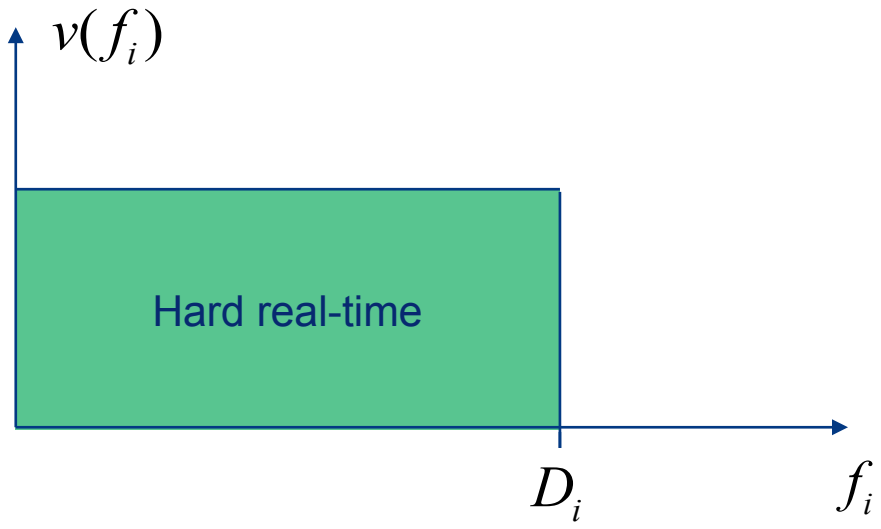


Performance metrics

Cost function – a general real-time performance metric

Cumulative value: $C = \sum_{\tau_i \in \mathbf{T}} v(f_i)$

Value associated with a task as a function of its completion time



Evaluating a real-time system

Recall from an earlier lecture that:

A scheduling algorithm is said to be optimal with respect to schedulability if it can always find a feasible schedule whenever any other scheduling algorithm can do so.

How do we compare the “capability of schedulability” of two scheduling algorithms?

Comparing schedulability

Consider two scheduling algorithms, A and B:

- A dominates B
  - If B can generate a feasible schedule for some task set, then A can also always generate a feasible schedule for the same task set, but **not necessarily** vice versa.
- A is equivalent to B
  - If B can generate a feasible schedule for some task set, then A can also generate a feasible schedule for the same task set, and **also** vice versa.
- A is incomparable to B
  - If none dominates the other **and** they are not equivalent.

Comparing schedulability

Consider two scheduling algorithms, A and B:

- Some conclusions:
  - If both A and B are optimal scheduling algorithm, then they are equivalent.
  - If A is optimal but B is not optimal, then A dominates B.
  - If neither A nor B are optimal, then we cannot conclude.

Schedulability analysis

Schedulability analysis:

The process of determining whether a task set can be scheduled by a given run-time scheduler in such a manner that all task instances will complete by their deadlines.

Schedulability analysis typically involves a feasibility test that is customized for the actual run-time scheduler used.

Schedulability analysis

Complexity of uniprocessor schedulability analysis:
(Baruah et al, 1990)

The problem of deciding if a task set can be scheduled on one processor so that all task instances will complete by their deadlines is NP-hard in the strong sense.

Complexity of multiprocessor schedulability analysis:
(Leung & Whitehead, 1982)

The problem of deciding if a task set can be scheduled on m processors is NP-complete in the strong sense.

Schedulability analysis

Main aspects of schedulability analysis:

- The priority assignment problem
  - Given a set of tasks, *does there exist an assignment of priorities to these tasks* satisfying the property that the system can be scheduled by a priority-based run-time system such that all task instances will complete by their deadlines?
- The feasibility testing problem
  - Given a set of tasks, *and an assignment of priorities to these tasks*, can the system be scheduled by a priority-based run-time system such that all task instances will complete by their deadlines?

Schedulability analysis

Complexity of feasibility testing:

(Leung, 1989; Baruah et al 1990)

The problem of deciding the feasibility of a schedule produced on m ≥ 1 processors by a particular static or dynamic priority assignment is NP-hard in the strong sense.

Observation:

- If an optimal priority assignment can be found in polynomial time, the complexity of schedulability analysis reduces to that of the feasibility testing problem.

Priority assignment

A priority assignment policy *P* is said to be optimal with respect to a feasibility test *S* and a given task model, if and only if the following holds: *P* is optimal if there are no task sets that are compliant with the task model that are deemed schedulable by test *S* using another priority assignment policy, that are not also deemed schedulable by test *S* using policy *P*.

Observations:

- The definition is applicable to both sufficient feasibility tests and exact feasibility tests; optimal performance is still provided with respect to the limitations of the test itself.

Priority assignment

Relaxing the zero offset assumption:

- In a *synchronous* task set the offsets of tasks are identical, that is: $\forall i, j : O_i = O_j$

  This is the assumption for RM, DM and EDF to be optimal for the single-processor case.

- In an *asynchronous* task set the offsets of at least one pair of tasks are not identical, that is: $\exists i, j : i \neq j, O_i \neq O_j$

  A priority-assignment policy that is shown to be optimal for synchronous task sets is not necessarily optimal for asynchronous task sets.

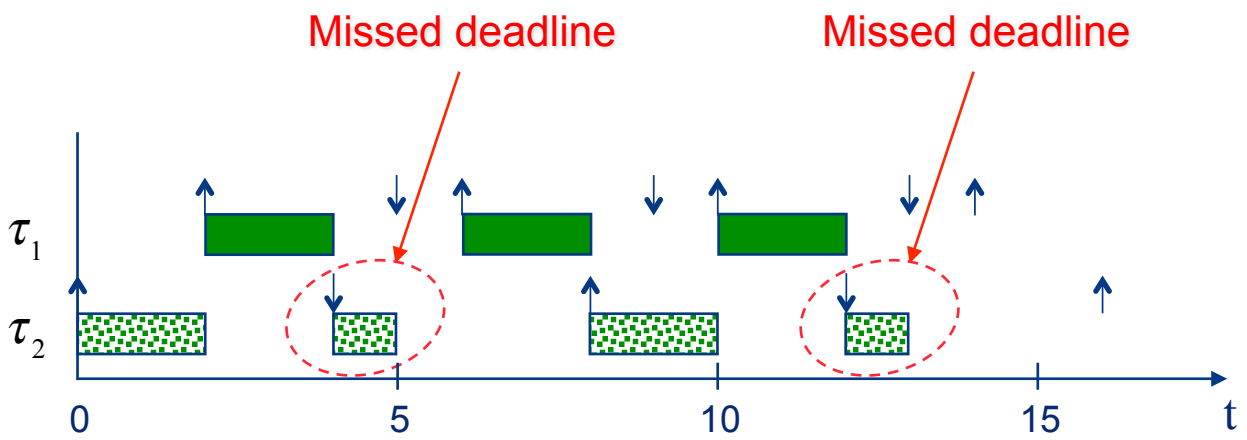
  For example, it is known that RM and DM are not optimal for asynchronous task systems. (Leung & Whitehead, 1982)

Priority assignment

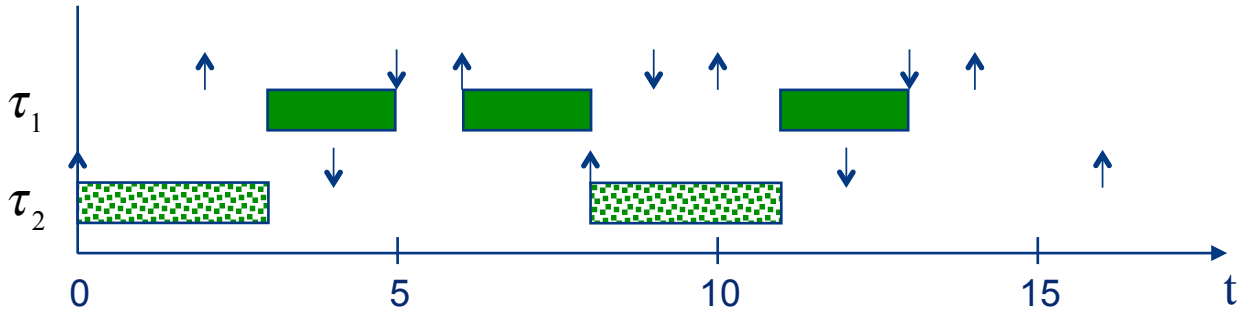
Non-optimality of DM for asynchronous tasks:

$\tau_i : (O_i, C_i, D_i, T_i)$
$\tau_1 : (2,2,3,4)$
$\tau_2 : (0,3,4,8)$

DM



Inverse DM

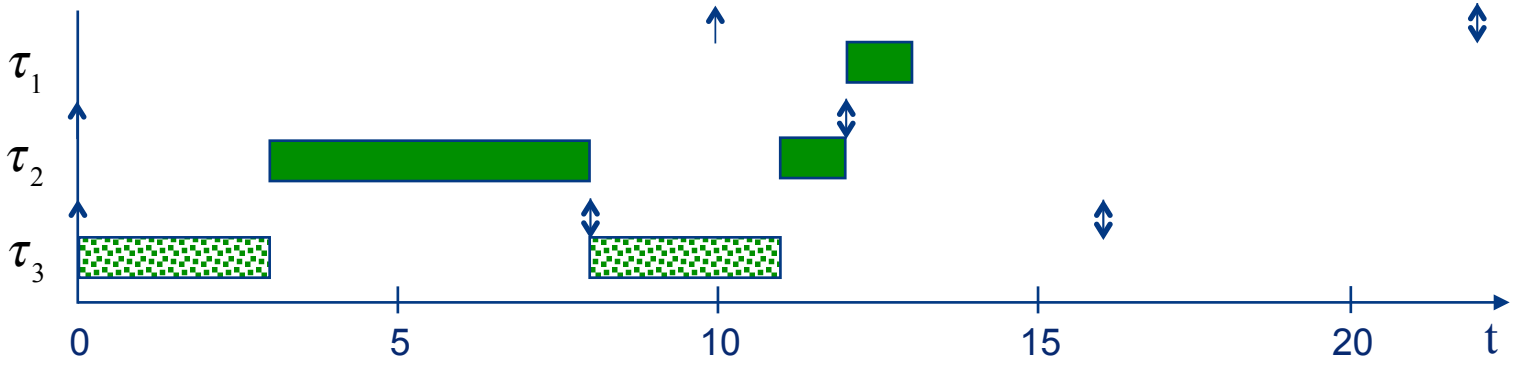


Priority assignment

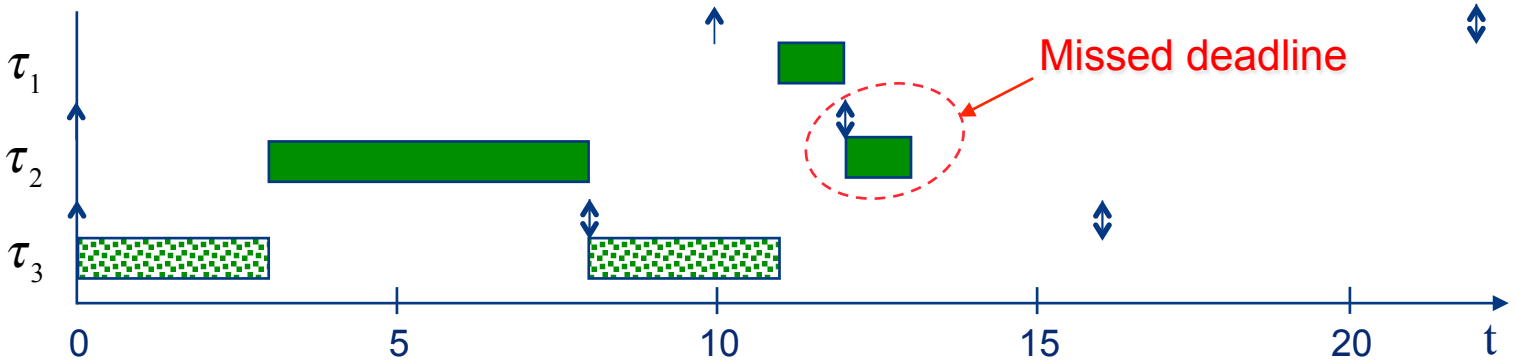
$\tau_i : (O_i, C_i, T_i)$
$\tau_1 : (10,1,12)$
$\tau_2 : (0,6,12)$
$\tau_3 : (0,3,8)$

Non-optimality of RM for asynchronous tasks:

RM



RM
(alternate tie-breaking rule)



Priority assignment

Complexity of uniprocessor schedulability analysis:

(Leung & Whitehead, 1982)

There exists a pseudo-polynomial time algorithm to decide if a synchronous task set can be scheduled using static priorities on one processor in such a way that all task instances will complete by their deadlines.

Proof:

- The deadline-monotonic priority assignment is optimal for synchronous task sets, and can be obtained in polynomial time
- An exact feasibility test for synchronous task sets on a single processor can be performed in pseudo-polynomial time (using response-time analysis).

Priority assignment

Complexity of uniprocessor schedulability analysis:

(Baruah et al, 1990)

There exists a pseudo-polynomial time algorithm to decide if a synchronous task set can be scheduled using dynamic priorities on one processor in such a way that all task instances will complete by their deadlines.

Proof:

- The earliest-deadline-first priority assignment is optimal for synchronous task sets, and can be obtained in polynomial time
- An exact feasibility test for synchronous task sets on a single processor can be performed in pseudo-polynomial time (using processor-demand analysis) if the total task utilization is < 1.

Priority assignment

Complexity of uniprocessor schedulability analysis:

(Baruah et al, 1990)

The problem of deciding if an asynchronous task set can be scheduled on one processor so that all task instances will complete by their deadlines is NP-hard in the strong sense.

Observations:

- If the tasks are ever simultaneously released (can be decided in pseudo-polynomial time), the synchronous case applies and schedulability can be decided in pseudo-polynomial time.
- If the tasks are never simultaneously released it is necessary to find an optimal priority assignment and an exact test for that priority assignment.

Priority assignment

Optimal Priority Assignment (OPA) algorithm: (Audsley, 1991)

1. The tasks are divided in two sets: an assigned set **A**, consisting of lower-priority tasks with given priorities, and an unassigned set **U** consisting of tasks with no given priority (but assumed to have higher priority than tasks in **A**). Initially, all tasks are in **U**.
2. All tasks in **U** are chosen in turn and temporarily placed as the highest-priority task in **A** and tested for schedulability.
3. If the chosen task is schedulable the tentative priority of the task is established, and the task is permanently moved to **A**. If the task is not schedulable it is returned to **U**.
4. This continues until either all tasks in **U** have been checked and found to be unschedulable, or all tasks have been moved to **A** and thus have the final priority assignment.

Priority assignment

OPA algorithm (Audsley, 1991)

```
for each priority level k, lowest first
{
   for each unassigned task τ
   {
      if τ is schedulable at priority k
      according to schedulability test S
      with all unassigned tasks assumed to
      have higher priorities
      {
         assign τ to priority k
         break (continue outer loop)
      }
   }
   return unschedulable
}
return schedulable
```

Priority assignment

Properties of the OPA algorithm:

- The time complexity of OPA is $O(n^2+n)$, for n tasks, …
  - This is significantly better than having to consider all $n!$ possible priority orderings.
  - … times the time complexity of the schedulability test.
- Optimality of the OPA algorithm is provided with respect to the limitations of the schedulability test used.
  - If a non-exact schedulability test is used the priority ordering will reflect the quality of the test.
- The OPA algorithm only works under certain assumptions.
  - For example, a task being assigned a higher priority cannot become unschedulable according to the schedulability test, if it was previously deemed schedulable at the lower priority.