CHALMERS | UNIVERSITY OF GOTHENBURG

Parallel & Distributed Real-Time Systems

Lecture #5

Risat Pathan

Department of Computer Science and Engineering
Chalmers University of Technology

CHALMERS | UNIVERSITY OF GOTHENBURG

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.

CHALMERS | UNIVERSITY OF GOTHENBURG

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.

CHALMERS | UNIVERSITY OF GOTHENBURG

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?

CHALMERS | UNIVERSITY OF GOTHENBURG

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 the priority assignment problem reduces to that of the feasibility testing problem.

CHALMERS | UNIVERSITY OF GOTHENBURG

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.

CHALMERS | UNIVERSITY OF GOTHENBURG

Priority assignment

Relaxing the zero offset assumption:

- In order for the RM, DM and EDF priority-assignment policies to be optimal for the single-processor case we assume *synchronous* task sets where the offsets of tasks are identical, that is:

  $$\forall i, j : O_i = O_j$$

  In *asynchronous* task sets the offsets of at least one pair of tasks are not identical, that is:

  $$\exists i, j : i \neq j, O_i \neq O_j$$

  Asynchronous task sets are typically used to reduce jitter or to remove the need for resource access protocols (e.g. PCP).

CHALMERS | UNIVERSITY OF GOTHENBURG

Priority assignment

Relaxing the zero offset assumption (cont'd):

- In an asynchronous task set two tasks with identical periods but different offsets could never be released simultaneously during the lifetime of the system.

  This means that the worst-case response times of the tasks will be lower than if the offsets of the task were equal.

- A priority-assignment policy that is shown to be optimal for a synchronous system is not necessarily optimal for an asynchronous system.

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

CHALMERS | UNIVERSITY OF GOTHENBURG

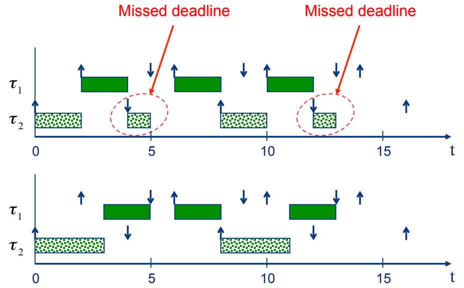
Priority assignment

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



CHALMERS | UNIVERSITY OF GOTHENBURG

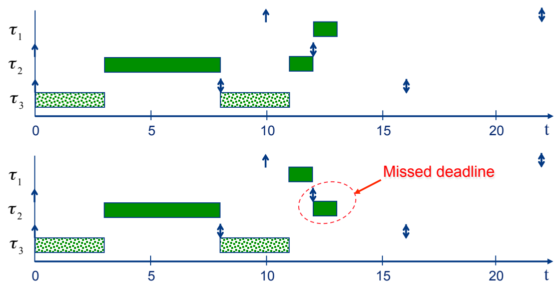
Priority assignment

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

Non-optimality of RM for asynchronous tasks:

RM

RM (alternate tie-breaking rule)



CHALMERS | UNIVERSITY OF GOTHENBURG

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).

CHALMERS | UNIVERSITY OF GOTHENBURG

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 task utilization is less than 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. A priority ordering is partitioned into two parts: a sorted part, consisting of the lower *n* priority tasks, and the remaining unsorted higher priority tasks. Initially the priority ordering is an arbitrary one, and all tasks are unsorted.
2. All tasks in the unsorted partition are chosen in turn and placed at the top of the sorted partition and tested for schedulability.
3. If the chosen task is schedulable then the priority of the task is left as it is, and the sorted partition extended by one position. If the task is not schedulable it is returned to its former priority.
4. This continues until either all tasks in the unsorted partition have been checked and found to be unschedulable, or else the sorted partition constitutes the final priority assignment.

Priority assignment

Optimal Priority Assignment Algorithm

```
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
```

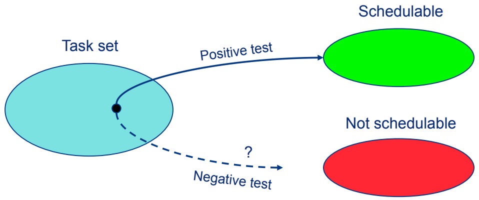

Priority assignment

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 reflects the quality of the test.
- The OPA algorithm holds for any scheduling test where a task being assigned a higher priority cannot become unschedulable according to the test, if it was previously deemed schedulable at the lower priority.

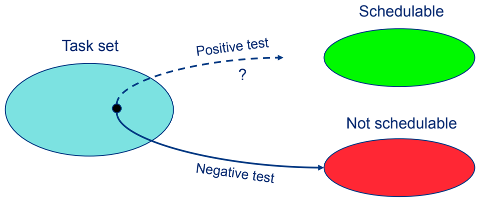
Feasibility testing

- A feasibility test is sufficient if it with a positive answer shows that a set of tasks is definitely schedulable.
  - A negative answer says nothing! A set of tasks can still be schedulable despite a negative answer.



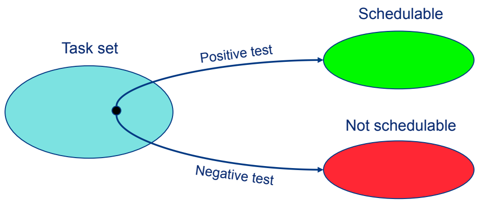
Feasibility testing

- A feasibility test is necessary if it with a negative answer shows that a set of tasks is definitely not schedulable.
  - A positive answer says nothing! A set of tasks can still be impossible to schedule despite a positive answer.



Feasibility testing

- An exact feasibility test is both sufficient and necessary. If the answer is positive the task set is definitely schedulable, and if the answer is negative the task set is definitely not schedulable.



Feasibility testing

What techniques for feasibility testing exist?

- Hyper-period analysis (for static and dynamic priorities)
  - In a simulated schedule no task execution may miss its deadline
- Guarantee bound analysis (for static and dynamic priorities)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (for static priorities)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (for dynamic priorities)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Feasibility testing

What techniques for feasibility testing exist?

- Hyper-period analysis (exponential time complexity)
  - In a simulated schedule no task execution may miss its deadline
- Guarantee bound analysis (polynomial time complexity)
  - The fraction of processor time that is used for executing the task set must not exceed a given bound
- Response time analysis (pseudo-polynomial complexity)
  - The worst-case response time for each task must not exceed the deadline of the task
- Processor demand analysis (pseudo-polynomial complexity)
  - The accumulated computation demand for the task set under a given time interval must not exceed the length of the interval

Hyper-period analysis

Motivation:

- When it is not obvious which feasibility analysis should be used for a particular task set it is always possible to generate a schedule by simulating the execution of the tasks, and then check schedulability for individual tasks.
  For example, this is currently the only way to perform exact feasibility tests on asynchronous task sets where tasks will never be released simultaneously.
- The schedule interval that is sufficient to investigate is related to the hyper-period of the task set, that is, the least-common-multiple (LCM) of the task periods.
  Thus, hyper-period analysis will in general have an exponential time complexity.

Hyper-period analysis

Feasibility intervals:

- For synchronous systems it is sufficient to investigate the interval $[0, P]$, where P is the hyper-period of the task set.
- For asynchronous systems with dynamic priorities it is sufficient to investigate the interval $[0, O_{\max} + 2P]$, where P is the hyper-period and $O_{\max}$ is the largest offset in the task set.
- For asynchronous systems with static priorities it is sufficient to investigate, for each task τ_i, the interval $[O_i, O_i + P_i]$, where P_i is the hyper-period of all tasks with priority higher than τ_i.

Guarantee bound analysis

Basic principle:

- If the accumulated utilization U of all tasks in the system does not exceed a guarantee bound, all timing constraints will be met.
- The guarantee bound U_{GB} is expressed as a fraction of the available processing capacity of the system.
  (= 100% multiplied by the number of processors)
- The utilization U_i of a task is expressed as the fraction of processing capacity used for executing the task.

Thus, guarantee bound analysis will have a polynomial time complexity

task utilization $= \frac{C_i}{T_i}$

accumulated utilization $= \sum_{i=1}^{n} \frac{C_i}{T_i}$

CHALMERS | UNIVERSITY OF GOTHENBURG

Guarantee bound analysis

A good guarantee bound …

… enables prediction of required processing capacity, e.g. # and speed of processors, of the hardware (when software is known)

… enables derivation of timing parameters, e.g. periods of tasks, in the software (when hardware implementation is known)

=

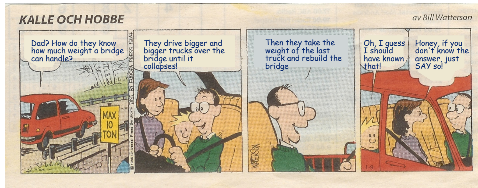
A good guarantee bound …

… enables prediction of how "strong" the hardware implementation must be (when the software "load" is known)

… enables prediction of how high the software "load" is allowed to be (when the "strength" of the hardware implementation is known)

CHALMERS | UNIVERSITY OF GOTHENBURG

Guarantee bound analysis



A good guarantee bound …

… enables prediction of how "strong" the hardware implementation must be (when the software "load" is known)

… enables prediction of how high the software "load" is allowed to be (when the "strength" of the hardware implementation is known)

CHALMERS | UNIVERSITY OF GOTHENBURG

Guarantee bound analysis

Guarantee bound analysis for RM: (Liu & Layland, 1973)

- The guarantee bound for RM scheduling is

$$U_{\text{GB-RM}} = n\left(2^{1/n} - 1\right)$$

- A conservative lower limit on the guarantee bound can be derived by letting $n \rightarrow \infty$

$$\lim_{n\to\infty} n\left(2^{1/n} - 1\right) = \ln 2 \approx 0.693$$

CHALMERS | UNIVERSITY OF GOTHENBURG

Guarantee bound analysis

Guarantee bound analysis for RM: (Liu & Layland, 1973)

- A sufficient condition for RM scheduling is

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

- The test is only valid if all of the following conditions apply:
  1. Single-processor system
  2. Synchronous task sets
  3. Independent tasks
  4. Periodic or sporadic tasks
  5. Tasks have deadlines equal to period ($D_i = T_i$)

CHALMERS | UNIVERSITY OF GOTHENBURG

Guarantee bound analysis

Guarantee bound analysis for RM: (Liu & Layland, 1973)

- The proof of the condition uses the fact that the worst-case response time for a task occurs at a critical instant (where the task arrives at the same time as all higher-priority tasks)
- The feasibility test is derived using an analysis of this special case
- The proof also shows that if the task set is schedulable for the critical instant case, it is also schedulable for any other case
- The proof is given in Krishna and Shin (Section 3.2.1) Highly recommended reading!

CHALMERS | UNIVERSITY OF GOTHENBURG

Guarantee bound analysis

Guarantee bound analysis for EDF: (Liu & Layland, 1973)

- A sufficient and necessary condition for EDF scheduling is

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

$$U_{\text{GB-EDF}} = 1$$

- The test is only valid if all of the following conditions apply:
  1. Single-processor system
  2. Synchronous task sets
  3. Independent tasks
  4. Periodic tasks
  5. Tasks have deadlines equal to period ($D_i = T_i$)