Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Real-Time Architectures 2003/2004

Scheduling Analysis

Johan Lukkien

19-04-2004

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Overview

- Algorithm and problem classes
- Simple, periodic taskset
  - problem statement
  - feasibility criteria
    - utilization bound
    - response time criterion
- Analysis of example algorithms
  - Rate Monotonic Scheduling
  - Earliest Deadline First
  - Deadline monotonic

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Issues, questions of interest

- Relevant properties of methods
  - cost functions, comparison, classification, optimality
- When to apply what method
  - criteria
    - system types, parameters
    - assumptions on execution environment
      - OS & platform
    - additional requirements (e.g. behavior under overload)
  - how to *prove* properties of a (method, taskset) combination
    - static and dynamic *tests* to demonstrate feasibility

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

System types

- Properties of the task set
  - periodic, sporadic tasks
  - fixed/dynamic parameters
  - deadline within period
  - precedence relations & preemptability
- Criticality mix
  - hard, firm and soft
    - hard and firm: acceptance test
    - overload possibilities
- Number and type of resources (e.g. processors)
- Modes: subsets of tasks
  - statically defined / dynamically created

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Algorithm classes

- Priority static/dynamic
  - fixed priority: priority of job fixed
  - dynamic task, fixed job priority
  - dynamic
- Preemptive
- Online / offline
  - online: e.g. admission/acceptance computation (guarantee), assignment of priorities
  - offline
    - precomputation of a table
    - complex optimizations possible
- Cost functions
  - e.g. maximum lateness, ...

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

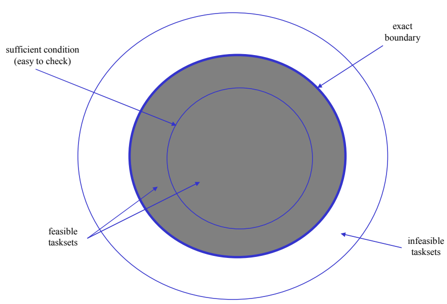
Overview

- Algorithm and problem classes
- Simple, periodic taskset
  - problem statement
  - feasibility criteria
    - utilization bound
    - response time criterion
- Analysis of example algorithms
  - Rate Monotonic Scheduling
  - Earliest Deadline First
  - Deadline monotonic

Problem

- System
  - periodic, preemtable taskset, *Z*
  - fixed parameters
  - deadline equal to period
  - non-blocking
  - no precedence relations
  - single processor
- Find
  - one or more algorithms that work
  - and for such an algorithm
    - a criterion for feasibility
    - an analysis method and proof
    - a comparison with other methods

Feasibility criteria



Utilization criterion

- Recall
  - $U_j = C_j / T_j$ utilization for task *j*
  - $U = \sum U_j$ total utilization
- Clearly,
  - for $U>1$ the set is not schedulable *by any algorithm* (overload)
    - **proof**: the amount of computation time in a hyperperiod *T* is the number of times each task releases a job times the computation time of that job, hence
    - $T \leq \sum C_j * T/T_j = T * \sum C_j / T_j$
    - with $U>1$, the latter term exceeds *T* which is a contradiction
  - for $U=0$ the set is schedulable (by any algorithm)
- A given utilization factor can be decreased by
  - decreasing computation times
  - increasing periods

Bounds

- Assume the algorithm is *independent of computation times*. These then can be varied to change the utilization factor.
- There is a utilization factor, dependent on algorithm and taskset such that computation times cannot be increased anymore without destroying feasibility
  - *Umax (Z, Alg)* -- increase computation times to the limit
- Minimizing over tasksets gives the *least utilization bound* for the algorithm
  - *U* (Alg)* = (**min** *Z*:: *Umax (Z, Alg)*)
- Meaning: for tasksets with utilizations below *U** the algorithm will produce a feasible schedule
  - *U*≤ *U**: acceptance criterion ('sufficient condition')

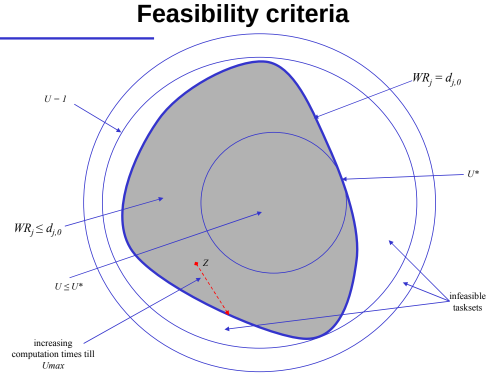
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Critical instant, WCRT

- A *critical instant* of a job $\tau_{j,i}$ is a combination of jobs (including $\tau_{j,i}$) with release times chosen such that $\tau_{j,i}$ has a worst-case response time.
- Worst-case response time (WCRT): determine *for the critical instant*, WR_j, the response time of the first job of task *j*.
- Then, WCRT feasibility criterion:
  - $WR_j \leq d_{j,0}$

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Feasibility criteria



Overview

- Algorithm and problem classes
- Simple, periodic taskset
  - problem statement
  - feasibility criteria
    - utilization bound
    - response time criterion
- Analysis of example algorithms
  - Rate Monotonic Scheduling
  - Earliest Deadline First
  - Deadline monotonic

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Rate Monotonic Scheduling

- Algorithm
  - fixed task priorities: higher priorities for shorter periods
  - preemptive
  - can be used both off- and online
  - minimize maximum lateness
- Advantages of RMS
  - simple, fixed priority assignment
  - in-depth analysis available
  - OS support
  - deals reasonable with overload conditions

- **NOTE:**
  - for now, assume tasks are sorted in order of increasing period

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

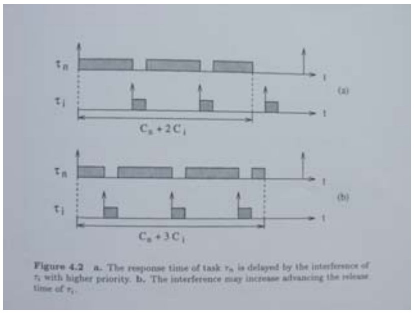
RMS (cnt'd)

- For RMS:
  - a critical instant of job $\tau_{j,i}$ occurs when $\tau_{j,i}$ is released simultaneously with all higher priority jobs

- Questions
  - *U*(RMS)* ?
  - *U*(RMS)* maximal in some sense?
  - what if *U*<U<1* ?
  - what about the response time of RMS?

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Critical Instant in RMS



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Example: utilization and schedulability

- Two tasks (written as *(C,T))*
  - (3,6) and (4,9)
  - RMS yields deadline miss at *t = 9*
  - *U* = 51/54 = 0.944

  - (2,4) and (4,8)
  - is feasible with RMS
  - *U = 1*

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Utilization bound for RMS

- $U^*(RMS) = n(2^{1/n}-1)$, *n* tasks
  - result due to Liu & Layland (called LL-bound)
  - converges to *ln(2)* (+/- 0.69)
    - hence, in fact *U*(RMS)* = *ln(2)* [independent of taskset]
  - worst case taskset: tricky

Example with WCRT

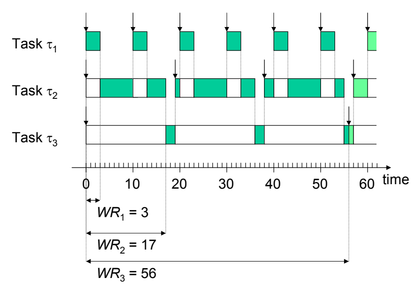
- Task set Z consisting of 3 tasks:

Task	Period T_j	Execution time C_j	Utilization U_j
τ_1	10	3	0.3
τ_2	19	11	0.58
τ_3	56	5	0.09

- **Notes**:
  - $U = 0.97 \leq 1$, hence Z *could* be schedulable;
  - $U_1+U_2 = 0.88 > LL(2) \approx 0.83$, therefore $U > LL(3)$, hence another test required.

Techniques

- **Time line**:

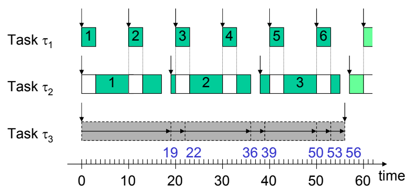


Iterative definition of WCRT

- Define a series of approximations as follows
  - $WR_j^{(0)} = C_j + (\sum i: 1 \leq i < j: C_i)$
  - $WR_j^{(k+1)} = C_j + (\sum i: 1 \leq i < j: \lceil WR_j^{(k)} / T_i \rceil C_i)$
- Remarks
  - the extra term represents the contribution of higher priority jobs in that time span
    - the interference
  - the procedure stops if $WR_j^{(k+1)} = WR_j^{(k)}$; the value then is the response time
  - termination is guaranteed because $U \leq 1$

Techniques

- **Calculation** (visualization):



$WR_3^{(6)} = WR_3^{(7)} = 5 + 6\cdot 3 + 3\cdot 11 = 56$

Recursive definition of WCRT

- Define:
  - *R(t, s, j): "response time at time t when s-t equals the computation time left to do for job $\tau_{j,i}$"*
  - .... then $WR_j = R(0, C_j, j)$
  - $Act(t,s,i) = \lceil s/T_i \rceil - \lceil t/T_i \rceil$
    - the number of activations of task *i* in the interval [t,s)
  - $Comp(t,s,j) = (\sum i\colon 1 \le i < j\colon Act(t,s,i) * C_i)$
    - the total time associated with interference from higher priority tasks in the interval [t,s)
- *R(t,s,j)* =
  - *s,*   *if Comp(t,s,j) = 0*
  - *R(s, s+Comp(t,s,j), j)*   *if Comp(t,s,j) ≠ 0*

Optimality

- RMS is *optimal* among all fixed priority preemptive algorithms
  - if a taskset can be scheduled feasibly with any fpp algorithm it can so with RMS

Earliest Deadline First

- Algorithm
  - job with nearest absolute deadline gets highest priority
  - minimizes maximum lateness
  - online algorithm
- Advantages / disadvantages
  - relatively simple
  - needs priority queue for storing deadlines
    - logarithmic access
  - needs dynamic priorities
  - behaves badly under overload