Real-Time Architectures  
2003/2004  

Scheduling Analysis  

Johan Lukkien

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

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
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

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, ....

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

- sufficient condition
  - (easy to check)
  - exact boundary

Feasible tasksets

Infeasible tasksets

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
      - $T \sum C_j / T_j = \sum C_j / U_j$
      - 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
  - \( U_{\text{max}}(Z, \text{Alg}) \) -- increase computation times to the limit
- Minimizing over tasksets gives the least utilization bound for the algorithm
  - \( U^*(\text{Alg}) = (\min Z: U_{\text{max}}(Z, \text{Alg})) \)
- Meaning: for tasksets with utilizations below \( U^* \) the algorithm will produce a feasible schedule
  - Use \( U^* \) acceptance criterion ('sufficient condition')

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:
  - \( WR_j \leq d_{j,0} \)

Feasibility criteria

- \( U = 1 \) infeasible tasksets
- Increasing computation times till \( U_{\text{max}} \)
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

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

RMS (cnt’d)

- For RMS:
  - a critical instant of job \( \tau_j \) occurs when \( \tau_j \) 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?
Critical Instant in RMS

Example: utilization and schedulability

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

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

Utilization bound for RMS

- \(U^*(RMS) = n(2^{ln-1})\), \(n\) tasks
  - result due to Liu & Layland (called LL-bound)
  - converges to \(ln(2) \approx 0.69\)
  - hence, in fact \(U^*(RMS) = ln(2)\) (independent of taskset)
  - worst case taskset: tricky
Example with WCRT

- Task set $Z$ consisting of 3 tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>Period $T_j$</th>
<th>Execution time $C_j$</th>
<th>Utilization $U_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>10</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>19</td>
<td>11</td>
<td>0.58</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>56</td>
<td>5</td>
<td>0.09</td>
</tr>
</tbody>
</table>

- 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
  - $WR^{(0)}_j = C_j + (\frac{\sum_{i: 1 \leq i < j} C_i}{T_j})$
  - $WR^{(k+1)}_j = C_j + (\frac{\sum_{i: 1 \leq i < j} WR^{(k)}_i / T_i}{C_j})$

- Remarks
  - the extra term represents the contribution of higher priority jobs in that time span
  - the interference
  - the procedure stops if $WR^{(k+1)}_j = WR^{(k)}_j$; the value then is the response time
  - termination is guaranteed because $U \leq 1$
Techniques

• Calculation (visualization):

![Diagram showing task scheduling and response times](image)

- Task $\tau_1$
- Task $\tau_2$
- Task $\tau_3$

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 $\tau_j$"
  - $\text{Comp}(t, s, j) = \sum_{i: 1 \leq i < j} \text{Act}(t, s, i) \cdot C_i$
  - $\text{Act}(t, s, i) = \left\lceil \frac{s}{T_i} \right\rceil - \left\lceil \frac{t}{T_i} \right\rceil$
  - $\text{Act}(t, s, i)$: the number of activations of task $i$ in the interval $[t, s)$

- $R(t, s, j) =$
  - $s$, if $\text{Comp}(t, s, j) = 0$
  - $R(s, s + \text{Comp}(t, s, j))$, if $\text{Comp}(t, s, j) \neq 0$

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