Real-time Scheduling of Periodic Tasks (1)

Advanced Operating Systems
Lecture 2
Lecture Outline

• Scheduling periodic tasks
• The rate monotonic algorithm
  • Definition
  • Non-optimality
  • Time-demand analysis
  • ...

Scheduling Periodic Tasks

- Simplest real-time system: a set of $n$ periodic tasks characterised by $T_i = (\phi_i, p_i, e_i, D_i)$ for $i = 1, 2, \ldots, n$
  - Simplified model: $T_i = (p_i, e_i)$ when $\phi_i = 0$ and $D_i = p_i$
  - Tasks are independent, with no resource constraints
  - Assume a single processor system
  - There are no aperiodic or sporadic tasks

- Must schedule system to ensure all deadlines met
  - What type of job scheduler is used?
  - What scheduling algorithm is used?
  - How to prove correctness of the schedule?
Job Schedulers

- **Clock driven scheduler**
  - Decisions on what job execute made at specific time instants
    - Usually regularly spaced, implemented using a periodic timer interrupt: scheduler awakes after each interrupt, schedules the job to execute for the next period, then sleeps until the next interrupt
    - E.g. the furnace control example, with an interrupt every 100ms
  - Primarily used for static systems, with schedule computed at design time and encoded as a fixed table
  - Behaviour depends on algorithm used to assign jobs to time slots

- **Priority driven scheduler**
  - Scheduler chooses what to run at job release or completion time, based on some notion of job priority; will always run a job, if one available
  - Flexible, as scheduling decisions made at runtime, but hard to validate
  - Behaviour depends on algorithm used to assign priorities to jobs
Scheduling Algorithms

• Wide range of scheduling algorithms used:
  • Rate monotonic (RM)
  • Deadline monotonic (DM)
  • Earliest deadline first (EDF)
  • Least slack time (LST)

• Trade-off optimality, stability, and ease of validation
### Proving Correctness of a Schedule

- A set of periodic tasks repeats after hyper-period, $H = \text{lcm}(p_i)$ for $i = 1, 2, \ldots, n$
- If the system can be scheduled for one hyper-period, it can be scheduled for all, given no aperiodic or sporadic tasks, and no resource constraints
- Can demonstrate correctness of schedule by exhaustive simulation, or using a mathematical proof of correctness

\[ H = \text{lcm}(3, 5) = 15 \]

\begin{itemize}
  
  - $T_1 : p_1 = 3, e_1 = 1$
  
  - $T_2 : p_2 = 5, e_2 = 2$
\
\end{itemize}
The Rate Monotonic Algorithm

- Assign priorities to jobs in each task based on the period of that task
  - Shorter period → higher priority; rate (of job releases) is the inverse of the period, so jobs with higher rate have higher priority
  - Rationale: schedule jobs with most deadlines first, fit others around them
  - All jobs in a task have the same priority – fixed priority algorithm

- For example, consider a system of 3 tasks:
  - $T_1 = (4, 1)$ ⇒ rate = $1/4$
  - $T_2 = (5, 2)$ ⇒ rate = $1/5$
  - $T_3 = (20, 5)$ ⇒ rate = $1/20$
  - Relative priorities: $T_1 > T_2 > T_3$
Rate Monotonic: Example

<table>
<thead>
<tr>
<th>Time</th>
<th>Ready to run</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(J_{2,1}, J_{3,1})</td>
<td>(J_{1,1})</td>
</tr>
<tr>
<td>1</td>
<td>(J_{3,1})</td>
<td>(J_{2,1})</td>
</tr>
<tr>
<td>2</td>
<td>(J_{3,1})</td>
<td>(J_{2,1})</td>
</tr>
<tr>
<td>3</td>
<td>(J_{3,1})</td>
<td>(J_{3,1})</td>
</tr>
<tr>
<td>4</td>
<td>(J_{3,1})</td>
<td>(J_{1,2})</td>
</tr>
<tr>
<td>5</td>
<td>(J_{3,1})</td>
<td>(J_{2,2})</td>
</tr>
<tr>
<td>6</td>
<td>(J_{3,1})</td>
<td>(J_{2,2})</td>
</tr>
<tr>
<td>7</td>
<td>(J_{3,1})</td>
<td>(J_{3,1})</td>
</tr>
<tr>
<td>8</td>
<td>(J_{3,1})</td>
<td>(J_{1,3})</td>
</tr>
<tr>
<td>9</td>
<td>(J_{3,1})</td>
<td>(J_{3,1})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Ready to run</th>
<th>Running</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(J_{3,1})</td>
<td>(J_{2,3})</td>
</tr>
<tr>
<td>11</td>
<td>(J_{3,1})</td>
<td>(J_{2,3})</td>
</tr>
<tr>
<td>12</td>
<td>(J_{3,1})</td>
<td>(J_{1,4})</td>
</tr>
<tr>
<td>13</td>
<td>(J_{3,1})</td>
<td>(J_{3,1})</td>
</tr>
<tr>
<td>14</td>
<td>(J_{3,1})</td>
<td>(J_{3,1})</td>
</tr>
<tr>
<td>15</td>
<td>(J_{2,4})</td>
<td>(J_{2,4})</td>
</tr>
<tr>
<td>16</td>
<td>(J_{2,4})</td>
<td>(J_{1,5})</td>
</tr>
<tr>
<td>17</td>
<td>(J_{2,4})</td>
<td>(J_{2,4})</td>
</tr>
<tr>
<td>18</td>
<td>(J_{2,4})</td>
<td>(empty)</td>
</tr>
</tbody>
</table>

All tasks meet deadlines: proof by exhaustive simulation
Low priority tasks (e.g., \(T_3\)) frequently preempted

\(T_1 = (4, 1)\)
\(T_2 = (5, 2)\)
\(T_3 = (20, 5)\)
Rate Monotonic is Not Optimal

- **Proof by counter-example:**
  - A system of two independent periodic tasks, $T_1 = (2, 1)$ and $T_2 = (5, 2.5)$, scheduled preemptively on a single processor:
  - $H = 10$, $U = 1.0 \rightarrow$ there is no slack time
  - Rate monotonic fails to meet deadlines, EDF (discussed later) succeeds
Time Demand Analysis

- Exhaustive simulation error prone and tedious – an alternative is *time demand analysis*
  - Fixed priority algorithms predictable; do not suffer *scheduling anomalies*
    - The worst case execution time of the system occurs with the worst case execution time of the jobs, unlike dynamic priority algorithms which can exhibit anomalous behaviour
  - Basis of general proof that system can be scheduled to meet all deadlines
    - Find *critical instants* when system is most loaded, and has its worst response time
    - Use time demand analysis to check if system can be scheduled at those instants
    - In absence of scheduling anomalies, system will meet all deadlines if it can be scheduled at critical instants
Finding Critical Instants

• Critical instant of a job is the worst-case release time for that job, taking into account all jobs that have higher priority
  
  • i.e., job is released at the same instant as all jobs with higher priority are released, and must wait for all those jobs to complete before it executes
  
  • Response time, $w_{i,k}$, of a job released at a critical instant is the maximum possible response time of that job

• Definition of a critical instant:

  \[
  \text{if } w_{i,k} \leq D_{i,k} \text{ for every } J_{i,k} \text{ in } T_i \text{ then} \\
  \quad \text{The job released at that instant has maximum response time of all jobs in } T_i \text{ and } W_i = w_{i,k} \\
  \text{else if } \exists J_{i,k} : w_{i,k} > D_{i,k} \text{ then} \\
  \quad \text{The job released at that instant has response time } > D \\
  \text{where } w_{i,k} \text{ is the response time of the job}
  \]

All jobs meet deadlines, but this is when job with the slowest response is started

If some jobs don’t meet deadlines, this is one of those jobs
Finding Critical Instants: Example

- 3 tasks scheduled using the rate-monotonic algorithm
- Response times of jobs in $T_2$ are: $r_{2,1} = 0.8$, $r_{2,2} = 0.2$, $r_{2,3} = 0.2$, $r_{2,4} = 0.2$, $r_{2,5} = 0.8$, ...
- Therefore critical instants of $T_2$ are $t = 0$ and $t = 10$
Time-demand Analysis

- Simulate system behaviour at the critical instants
  - For each job $J_{i,c}$ released at a critical instant, if $J_{i,c}$ and all higher priority tasks complete executing before their relative deadlines the system can be scheduled
  - Compute the total demand for processor time by a job released at a critical instant of a task, and by all the higher-priority tasks, as a function of time from the critical instant; check if this demand can be met before the deadline of the job:
    - Consider one task, $T_i$, at a time, starting highest priority and working down to lowest priority
    - Focus on a job, $J_i$, in $T_i$, where the release time, $t_0$, of that job is a critical instant of $T_i$
    - At time $t_0 + t$ for $t \geq 0$, the processor time demand $w_i(t)$ for this job and all higher-priority jobs released in $[t_0, t]$ is:
      $$w_i(t) = e_i + \sum_{k=1}^{i-1} \left\lfloor \frac{t}{p_k} \right\rfloor e_k$$
      
      *$w_i(t)$ is the time-demand function*
Using the Time-demand Function

• Compare time-demand function, \( w_i(t) \), and available time, \( t \):

  • If \( w_i(t) \leq t \) at some \( t \leq D_i \), the job, \( J_i \), meets its deadline, \( t_0 + D_i \)

  • If \( w_i(t) > t \) for all \( 0 < t \leq D_i \) then the task probably cannot complete by its deadline; and the system likely cannot be scheduled using a fixed priority algorithm

    • Note that this is a sufficient condition, but not a necessary condition. Simulation may show that the critical instant never occurs in practice, so the system could be feasible...

• Use this method to check that all tasks are can be scheduled if released at their critical instants; if so conclude the entire system can be scheduled
Time-demand Analysis: Example

The time-demand, $w_i(t)$, is a staircase function with steps at multiples of higher priority task periods.

Plot the time-demand versus available time graphically, to get intuition into approach.

Example: a rate monotonic system $T_1 = (3, 1), T_2 = (5, 2), T_3 = (10, 2)$ $U = 0.933$

Time-demand functions $w_1(t), w_2(t)$ and $w_3(t)$ are below $t$ at deadlines, so the system can be scheduled – simulate the system to check this!
Time-demand Analysis

• Works for any fixed-priority scheduling algorithm with periodic tasks where $D_i < p_i$ for all tasks

• Only a sufficient test:
  • System can be scheduled if time demand less than time available before critical instants
  • But, might be possible to schedule if time demand exceeds available time – further validation (i.e., exhaustive simulation) needed in this case
Summary

• The real-time scheduling problem for periodic tasks

• The rate monotonic algorithm
  • Simple, fixed-priority, algorithm
  • Non-optimal
  • Proofs of correctness of a schedule using exhaustive simulation and time-demand analysis

• Next lecture:
  • Alternative proofs of correctness for rate monotonic schedules
  • Other algorithms: deadline monotonic, earliest deadline first, least slack time