Periodic Task Scheduling

Radek Pelánek
Examples of Periodic Tasks

- sensory data acquisition
- control loops
- action planning
- system monitoring
Simplifying Assumptions

- constant period $T_i$
- all instances (jobs) of a task have the same computation time $C_i$
- no precedence relations, no resources
- preemption
- (deadline is equal to period $D_i = T_i$)
- (no aperiodic jobs)
Introduction

Periodic Scheduling

Aperiodic Jobs in Priority-driven Systems

Motivation and Assumptions

Example

<table>
<thead>
<tr>
<th>( C_i )</th>
<th>( \tau_1 )</th>
<th>( \tau_2 )</th>
<th>( \tau_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_i )</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

- find schedule
- think about possible scheduling algorithms
Outlook

- notions: jitter, processor utilization, schedulable utilization
- three basic approaches: static scheduling, dynamic priorities (EDF), fixed priorities (rate monotonic)
- examples
- discussion
Jitter

- deviation of the start/finishing time of consecutive instances of some task
- relative, absolute jitter
- for some applications it is important to minimize the jitter
- we do not deal with this issue in detail
Processor Utilization Factor

Definition (Processor utilization factor)

Given a set \( \Gamma \) of \( n \) periodic tasks, processor utilization factor \( U \) is the fraction of processor time spent in the execution of the task set:

\[
U = \sum_{i=1}^{n} \frac{C_i}{T_i}
\]

Example: \( U = \frac{2}{6} + \frac{4}{10} + \frac{3}{12} = \frac{59}{60} \)

Note: \( U > 1 \Rightarrow \) not schedulable
Schedulable Utilization

**Definition (Schedulable Utilization)**

schedulable utilization $U_S$ of a scheduling algorithm – the algorithm can feasibly schedule any set of periodic tasks with the total utilization of the tasks is $\leq U_S$

used to easily verify the schedulability of a task set
Cyclic Scheduling

- an approach, rather than algorithm
- (timeline scheduling, clock-driven scheduling)
- **static schedule**, constructed off-line
- schedule specifies exactly when each job executes
- *minor cycle* = greatest common divisor of periods
- *major cycle* = time after which the schedule repeats itself
Cyclic Scheduling

Example

\[ T_A = 25, \ T_B = 50, \ T_C = 100 \]
Aperiodic Jobs

- spare capacities in the static schedule can be used for handling aperiodic jobs
- aperiodic jobs can be scheduled e.g. by deadline based algorithm
Advantages and Disadvantages

- **advantages:**
  - simple, efficient (precomputed)
  - can deal with complex requirements, precedence constraints, ...
  - special requirements can be taken into account (e.g., minimizing jitter or context switches)

- **disadvantages:**
  - inflexible, difficult to modify and maintain
  - fragile (overrun may cause whole schedule to fail)
  - not very suitable for systems with both periodic and aperiodic tasks

Suitable for systems which are rarely modified once built (e.g., small embedded controllers).
Earliest Deadline First

- dynamic priority assignment
- selects tasks according to absolute deadline
- does not depend on periodicity; can be directly used for periodic + aperiodic tasks
Schedulability Analysis

Schedulable utilization of EDF is 1.

**Theorem**

*A set of periodic tasks is schedulable with EDF if and only if*

\[
\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1
\]
Extensions

- deadlines less than periods, aperiodic jobs
- the algorithm works directly for both extensions
- schedulability analysis is more complex (not covered)
Rate Monotonic Scheduling

- **priority based** algorithm: tasks scheduled according to priorities
- **fixed-priority** assignment: priorities assigned before the execution (all jobs of one task have the same priority)
- **priorities according to periods**: shorter period - higher priority
- **preemptive**
### Example

<table>
<thead>
<tr>
<th></th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$T_i$</td>
<td>6</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>
in general RM is not optimal

RM is optimal among fixed-priority algorithms

**Theorem**

*If a task set can be scheduled by fixed-priority algorithm then it can be scheduled by Rate Monotonic algorithm.*
Schedulable Utilization

For arbitrary set of periodic tasks, the schedulable utilization of the RM scheduling algorithm is

\[ U_S = n \left(2^{1/n} - 1\right) \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U_S )</td>
<td>1.00</td>
<td>0.82</td>
<td>0.78</td>
<td>0.76</td>
<td>0.74</td>
</tr>
</tbody>
</table>

For high values of \( n \) the schedulable utilization converges to

\[ U_{lub} = \ln 2 \sim 0.69 \]
a set of periodic tasks is simply periodic if for every pair of tasks: $T_i < T_j \Rightarrow T_j$ is an integer multiple of $T_i$

**Theorem**

A system of simply periodic tasks is schedulable according to the RM algorithm if and only if its utilization factor is $\leq 1$. 
Deadline Monotonic

- deadlines less than period
- priorities assigned according to inverse of relative deadlines
Example

<table>
<thead>
<tr>
<th></th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$T_i$</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

- What is the utilization factor?
- Is the task set schedulable?
- What is the schedule produced by EDF/RM?
Example

\[ U = \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \sim 0.97 \]
Example 2

<table>
<thead>
<tr>
<th></th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$T_i$</td>
<td>6</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

- What is the utilization factor? Is the task set schedulable?
- What is the schedule produced by RM?
Example 3

<table>
<thead>
<tr>
<th>$C_i$</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$T_i$</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

- What is the utilization factor? Is the task set schedulable?
- What is the schedule produced by RM/EDF?
## Comparison: RM vs EDF

<table>
<thead>
<tr>
<th></th>
<th>RM</th>
<th>EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>implementation</td>
<td>multi-level priority queue, $O(1)$</td>
<td>heap, $O(\log n)$</td>
</tr>
<tr>
<td>processor utilization</td>
<td>guarantee only for 0.69, practice 0.88</td>
<td>full utilization</td>
</tr>
<tr>
<td>context switches</td>
<td>many</td>
<td>few</td>
</tr>
<tr>
<td>guarantee test</td>
<td>nontrivial</td>
<td>simple</td>
</tr>
<tr>
<td>predictability</td>
<td>good</td>
<td>bad</td>
</tr>
</tbody>
</table>

In practice: fixed-priority schedulers
overload condition (processor utilization factor $> 1$), which tasks will not meet deadlines?

- EDF – unpredictable
- RM – predictable (tasks with the longest period)

reminder: Apollo 11 landing

- processor overload
- RM algorithm used $\Rightarrow$ predictable behaviour $\Rightarrow$ decision possible
real systems – combination of periodic and aperiodic tasks
main approaches:
  - **fixed priority servers**: scheduling of periodic tasks done by fixed priority algorithm (typically RM)
  - **dynamic priority servers**: scheduling of periodic tasks done by dynamic priority algorithm (typically EDF)
Assumptions and Remarks

- **periodic tasks** scheduled by a fixed priority algorithm (specifically rate monotonic)

- All periodic tasks start simultaneously at time $t = 0$, deadline = period

- Arrival times of aperiodic tasks are unknown beforehand

- Preemption

- Goal: meet deadlines of periodic tasks, minimize response time of aperiodic tasks

- Ordering of aperiodic tasks not discussed (done by some aperiodic scheduling algorithm, we will use FIFO)
Example

Periodic jobs:

<table>
<thead>
<tr>
<th></th>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$T_i$</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Aperiodic jobs:

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>2</td>
<td>8</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>$C_i$</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Background Scheduling

- aperiodic tasks scheduled in **background** (when no periodic task is running)
- schedule of **periodic** tasks is **not changed**
- major problem: high periodic load $\Rightarrow$ poor response times for aperiodic tasks
Figure 5.1  Example of background scheduling of aperiodic requests under Rate Mono-
tonic.
Realization

Figure 5.2  Scheduling queues required for background scheduling.
Server for Aperiodic Tasks

- periodic task whose purpose is to service aperiodic requests
- period $T_S$, computation time $C_S$ (capacity)
- scheduled in the same way as periodic tasks
- note: selection of $T_S$, $C_S$ – total utilization factor must remain $\leq 1$
Polling Server

- the simplest variant of server
- when active: serve pending aperiodic requests within its capacity
- no aperiodic requests are pending $\Rightarrow$ suspend
Polling Server: Example

Figure 5.3  Example of a Polling Server scheduled by RM.
Improving Polling Server

- how can we improve the performance?
- (consider the previous example)
Deferrable Server

- similar to polling server
- if no aperiodic requests are pending:
  - suspend itself
  - preserve capacity until the end of the period
  - if aperiodic request arrives later during the period: it is served
- at the beginning of the period capacity is fully replenished
Deferrable Server: Example

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$C_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Server

$C_s = 2$

$T_s = 5$

![Diagram](Image)

**Figure 5.5** Example of a Deferrable Server scheduled by RM.
Deferrable Server: Properties

- deferrable server provides better responsiveness than polling server
- schedulability analysis more complicated
  - deferrable server is not equivalent to periodic task
Deferrable Server: Example 2

Figure 5.7 DS is not equivalent to a periodic task. In fact, the periodic set \{τ₁, τ₂\} is schedulable by RM (a); however, if we replace τ₁ with DS, τ₂ misses its deadline (b).
Priority Exchange

- periodic server with high priority
- preserves capacity by exchanging it for the execution time of a lower-priority task:
  - at the beginning of the period: replenish the capacity
  - aperiodic requests are pending: serve them
  - no aperiodic requests are pending: exchange execution time with the active periodic task with the highest priority
- the priority exchange is performed repeatedly
Example

<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$C_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_2$</td>
<td>8</td>
<td>20</td>
</tr>
</tbody>
</table>

Server

$C_s = 1$

$T_s = 5$

**Figure 5.12** Example of aperiodic service under a PE server.
no periodic server; passive task *Slack Stealer*

slack = $d_i - t - c_i(t)$

main idea:
- no benefit in early completion of periodic tasks
- when aperiodic request arrives: steal available slacks

better responsiveness, more complicated schedulability analysis
Figure 5.20 Example of Slack Stealer behavior: a. when no aperiodic requests are pending; b. when an aperiodic request of three units arrives at time $t = 8$. 
Non-existence of Optimal Servers

Theorem

For any set of periodic tasks ordered on a given fixed-priority scheme and aperiodic requests ordered according to a given aperiodic queueing discipline, there does not exist any valid algorithm that minimizes the response time of every soft aperiodic request.

Similarly for average response time.
## Evaluation of Fixed Priority Servers

![Image showing evaluation of fixed-priority servers](image)

**Figure 5.26** Evaluation summary of fixed-priority servers.
Example

<table>
<thead>
<tr>
<th>periodic tasks</th>
<th>aperiodic tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>$J_1$</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>$J_2$</td>
</tr>
<tr>
<td>$C_i$</td>
<td>$J_3$</td>
</tr>
<tr>
<td>$T_i$</td>
<td>$C_i$</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Create schedules and determine response times:

- background scheduling
- polling server with intermediate priority
- deferrable server with the highest priority
Summary of the Lecture

- scheduling periodic tasks
  - cyclic scheduling (static schedule)
  - rate monotonic scheduling (static priorities)
  - earliest deadline first scheduling (dynamic priorities)

- processor utilization factor, schedulable utilization

- aperiodic tasks in periodic systems: fixed priority servers