Classical scheduling algorithms for periodic systems

Peter Marwedel
TU Dortmund, Informatik 12
Germany

2012年 12 月 19 日

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.
Structure of this course

Numbers denote sequence of chapters
Classes of mapping algorithms considered in this course

- **Classical scheduling algorithms**
  Mostly for independent tasks & ignoring communication, mostly for mono- and homogeneous multiprocessors

- **Dependent tasks as considered in architectural synthesis**
  Initially designed in different context, but applicable

- **Hardware/software partitioning**
  Dependent tasks, heterogeneous systems, focus on resource assignment

- **Design space exploration using evolutionary algorithms**; Heterogeneous systems, incl. communication modeling
Periodic scheduling

Each execution instance of a task is called a job.

Notion of optimality for aperiodic scheduling does not make sense for periodic scheduling.

For periodic scheduling, the best that we can do is to design an algorithm which will always find a schedule if one exists.

A scheduler is defined to be optimal iff it will find a schedule if one exists.
Periodic scheduling: Scheduling with no precedence constraints

Let \( \{T_i\} \) be a set of tasks. Let:

- \( p_i \) be the period of task \( T_i \),
- \( c_i \) be the execution time of \( T_i \),
- \( d_i \) be the **deadline interval**, that is, the time between \( T_i \) becoming available and the time until which \( T_i \) has to finish execution.
- \( l_i \) be the **laxity** or **slack**, defined as \( l_i = d_i - c_i \)
- \( f_i \) be the finishing time.

![Diagram of task availability]

Availability of Task \( i \quad -\quad -\quad -\quad \Rightarrow\) 

\( t \)

\( p_i \)

\( d_i \)

\( c_i \)

\( l_i \)
Average utilization: important characterization of scheduling problems:

Average utilization:

\[ \mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \]

Necessary condition for schedulability (with \( m=\)number of processors):

\[ \mu \leq m \]
Independent tasks: Rate monotonic (RM) scheduling

Most well-known technique for scheduling independent periodic tasks [Liu, 1973].

Assumptions:
- All tasks that have hard deadlines are periodic.
- All tasks are independent.
- $d_i = p_i$, for all tasks.
- $c_i$ is constant and is known for all tasks.
- The time required for context switching is negligible.
- For a single processor and for $n$ tasks, the following equation holds for the average utilization $\mu$:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$
Rate monotonic (RM) scheduling
- The policy -

RM policy: The priority of a task is a monotonically decreasing function of its period.
At any time, a highest priority task among all those that are ready for execution is allocated.

Theorem: If all RM assumptions are met, schedulability is guaranteed.
Maximum utilization for guaranteed schedulability

Maximum utilization as a function of the number of tasks:

$$\mu = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n(2^{1/n} - 1)$$

$$\lim_{n \to \infty} (n(2^{1/n} - 1)) = \ln(2)$$
Example of RM-generated schedule

$T_1$ preempts $T_2$ and $T_3$.

$T_2$ and $T_3$ do not preempt each other.
Failing RMS

Task 1: period 5, execution time 3
Task 2: period 8, execution time 3
\[ \mu = \frac{3}{5} + \frac{3}{8} = \frac{24}{40} + \frac{15}{40} = \frac{39}{40} \approx 0.975 \]
\[ 2(2^{1/2} - 1) \approx 0.828 \]
Case of failing RM scheduling

Task 1: period 5, execution time 2
Task 2: period 7, execution time 4
\[ \mu = \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \approx 0.97 \]
\[ 2(2^{1/2}-1) \approx 0.828 \]

Missed deadline
Missing computations scheduled in the next period
Intuitively: Why does RM fail?

No problem if $p_2 = m \cdot p_1$, $m \in \mathbb{N}$:

Switching to $T_1$ too early, despite early deadline for $T_2$.

fits
Critical instants

**Definition:** A **critical instant** of a task is the time at which the release of a task will produce the largest response time.

**Lemma:** For any task, the **critical instant** occurs if that task is simultaneously released with all higher priority tasks.

**Proof:** Let $T=\{T_1, \ldots, T_n\}$: periodic tasks with $\forall i: p_i \leq p_{i+1}$.

Critical instances (1)

Response time of $T_n$ is delayed by tasks $T_i$ of higher priority:

$T_n$  

$T_i$  

Stage duration: $c_n+2c_i$

Delay may increase if $T_i$ starts earlier

$T_n$  

$T_i$  

Stage duration: $c_n+3c_i$

Maximum delay achieved if $T_n$ and $T_i$ start simultaneously.
Critical instants (2)

Repeating the argument for all $i = 1, \ldots, n-1$:

- The worst case response time of a task occurs when it is released simultaneously with all higher-priority tasks.
  q.e.d.

- Schedulability is checked at the critical instants.
- If all tasks of a task set are schedulable at their critical instants, they are schedulable at all release times.
- Observation helps designing examples
The case \( \forall i: \quad p_{i+1} = m_i p_i \)

Lemma*: If each task period is a multiple of the period of the next higher priority task, then schedulability is also guaranteed if \( \mu \leq 1 \).

**Proof**: Assume schedule of \( T_i \) is given. Incorporate \( T_{i+1} \):

- \( T_{i+1} \) fills idle times of \( T_i \);
- \( T_{i+1} \) completes in time, if \( \mu \leq 1 \).

\[ T_i \]
\[ T_{i+1} \]
\[ T'_{i+1} \]

Used as the higher priority task at the next iteration.

* wrong in the book of 2003
Proof of the RM theorem

Let $T=\{T_1, T_2\}$ with $p_1 < p_2$.

Assume RM is not used $\implies$ prio($T_2$) is highest:

Schedule is feasible if $c_1 + c_2 \leq p_1$ \hspace{1cm} (1)

Define $F=\lfloor \frac{p_2}{p_1} \rfloor$: # of periods of $T_1$ fully contained in $T_2$
Case 1: $c_1 \leq p_2 - Fp_1$

Assume RM is used $\rightarrow$ prio($T_1$) is highest:

Case 1*: $c_1 \leq p_2 - Fp_1$
(c$_1$ small enough to be finished before 2nd instance of $T_2$)

Schedulable if $$(F + 1)c_1 + c_2 \leq p_2$$

*(Typos in [Buttazzo 2002]: $<$ and $\leq$ mixed up)*
Proof of the RM theorem (3)

Not RM: schedule is feasible if \( c_1 + c_2 \leq p_1 \) \hspace{1cm} (1)

RM: schedulable if \( (F+1)c_1 + c_2 \leq p_2 \) \hspace{1cm} (2)

From (1):
\[ Fc_1 + Fc_2 \leq Fp_1 \]

Since \( F \geq 1 \):
\[ Fc_1 + c_2 \leq Fc_1 + Fc_2 \leq Fp_1 \]

Adding \( c_1 \):
\[ (F+1)c_1 + c_2 \leq Fp_1 + c_1 \]

Since \( c_1 \leq p_2 - Fp_1 \):
\[ (F+1)c_1 + c_2 \leq Fp_1 + c_1 \leq p_2 \]

Hence: if (1) holds, (2) holds as well

For case 1: Given tasks \( T_1 \) and \( T_2 \) with \( p_1 < p_2 \), then if the schedule is feasible by an arbitrary (but fixed) priority assignment, it is also feasible by RM.
Case 2: $c_1 > p_2 - Fp_1$

(c_1 large enough not to finish before 2^{nd} instance of $T_2$)

Schedulable if

\[ Fc_1 + c_2 \leq Fp_1 \quad (3) \]
\[ c_1 + c_2 \leq p_1 \quad (1) \]

Multiplying (1) by $F$ yields

\[ Fc_1 + Fc_2 \leq Fp_1 \]

Since $F \geq 1$:

\[ Fc_1 + c_2 \leq Fc_1 + Fc_2 \leq Fp_1 \]

Same statement as for case 1.
Calculation of the least upper utilization bound

Let $T=\{T_1, T_2\}$ with $p_1 < p_2$.

Proof procedure: compute least upper bound $U_{lup}$ as follows

- Assign priorities according to RM
- Compute upper bound $U_{up}$ by setting computation times to fully utilize processor
- Minimize upper bound with respect to other task parameters

As before: $F = \lfloor p_2/p_1 \rfloor$

$c_2$ adjusted to fully utilize processor.
Case 1: $c_1 \leq p_2 - F p_1$

Largest possible value of $c_2$ is $c_2 = p_2 - c_1 (F + 1)$

Corresponding upper bound is

$$U_{ub} = \frac{c_1}{p_1} + \frac{c_2}{p_2} = \frac{c_1}{p_1} + \frac{p_2 - c_1 (F + 1)}{p_2} = \frac{1}{p_1} \cdot \frac{c_1}{p_1} - \frac{c_1 (F + 1)}{p_2} = 1 + \frac{c_1}{p_1} \left\{ \frac{p_2}{p_1} - (F + 1) \right\}$$

\{ \} is <0 $\rightarrow$ $U_{ub}$ monotonically decreasing in $c_1$

Minimum occurs for $c_1 = p_2 - F p_1$
Case 2: $c_1 \geq p_2 - Fp_1$

Largest possible value of $c_2$ is $c_2 = (p_1-c_1)F$

Corresponding upper bound is:

$$U_{ub} = \frac{c_1}{p_1} + \frac{c_2}{p_2} = \frac{c_1}{p_1} + \frac{(p_1-c_1)}{p_2} F = \frac{p_1}{p_2} F + \frac{c_1}{p_2} - \frac{c_1}{p_2} F = \frac{p_1}{p_2} F + \frac{c_1}{p_2} \left\{ \frac{p_2}{p_1} - F \right\}$$

\{ \} $\geq 0 \rightarrow U_{ub}$ monotonically increasing in $c_1$ (independent of $c_1$ if \{\}=0)

Minimum occurs for $c_1 = p_2 - Fp_1$, as before.
Utilization as a function of $G=p_2/p_1-F$

For minimum value of $c_1$:

$$U_{ub} = \frac{p_1}{p_2} F + \frac{c_1}{p_2} \left( \frac{p_2}{p_1} - F \right) = \frac{p_1}{p_2} F + \left( \frac{p_2 - p_1 F}{p_2} \right) \left( \frac{p_2}{p_1} - F \right) = \frac{p_1}{p_2} \left\{ F + \left( \frac{p_2}{p_1} - F \right) \left( \frac{p_2}{p_1} - F \right) \right\}$$

Let $G = \frac{p_2}{p_1} - F$;  \[ \Rightarrow \]

$$U_{ub} = \frac{p_1}{p_2} \left( F + G^2 \right) = \left( \frac{F + G^2}{p_2 / p_1} \right) = \left( \frac{F + G^2}{p_2 / p_1 - F} + F \right) = \left( \frac{F + G^2}{F + G} \right) = \frac{(F + G) - (G - G^2)}{F + G}$$

$$= 1 - \frac{G(1-G)}{F + G}$$

Since $0 \leq G < 1$:  \[ G(1-G) \geq 0 \]  \[ \Rightarrow \]  \[ U_{ub} \] increasing in $F$  \[ \Rightarrow \]  Minimum of $U_{ub}$ for $\min(F)$:  \[ F=1 \]  \[ \Rightarrow \]

$$U_{ub} = \frac{1 + G^2}{1 + G}$$
Proving the RM theorem for $n=2$

\[ U_{ub} = \frac{1+G^2}{1+G} \]

Using derivative to find minimum of $U_{ub}$:

\[ \frac{dU_{ub}}{dG} = \frac{2G(1+G) - (1+G^2)}{(1+G)^2} = \frac{G^2 + 2G - 1}{(1+G)^2} = 0 \]

\[ G_1 = -1 - \sqrt{2}; \quad G_2 = -1 + \sqrt{2}; \]

Considering only $G_2$, since $0 \leq G < 1$:

\[ U_{ub} = \frac{1 + (\sqrt{2} - 1)^2}{1 + (\sqrt{2} - 1)} = \frac{4 - 2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2} - 1) = 2(2^2 - 1) \approx 0.83 \]

This proves the RM theorem for the special case of $n=2$
Properties of RM scheduling

- RM scheduling is based on static priorities. This allows RM scheduling to be used in an OS with static priorities, such as Windows NT.
- No idle capacity is needed if $\forall i: p_{i+1} = F p_i$: i.e. if the period of each task is a multiple of the period of the next higher priority task, schedulability is then also guaranteed if $\mu \leq 1$.
- A huge number of variations of RM scheduling exists.
- In the context of RM scheduling, many formal proofs exist.
EDF can also be applied to periodic scheduling.

EDF optimal for every hyper-period
(= least common multiple of all periods)

- Optimal for periodic scheduling

- EDF must be able to schedule the example in which RMS failed.
Comparison EDF/RMS

RMS:

\[ T_1 \]

\[ T_2 \]

EDF:

\[ T_1 \]

\[ T_2 \]

\( T_2 \) not preempted, due to its earlier deadline.
EDF: Properties

EDF requires dynamic priorities

EDF cannot be used with an operating system just providing static priorities.

However, a recent paper (by Margull and Slomka) at DATE 2008 demonstrates how an OS with static priorities can be extended with a plug-in providing EDF scheduling (key idea: delay tasks becoming ready if they shouldn’t be executed under EDF scheduling.)
# Comparison RMS/EDF

<table>
<thead>
<tr>
<th>Priorities</th>
<th>RMS</th>
<th>EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Works with OS with fixed priorities</td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td>Uses full computational power of processor</td>
<td>No, just up till $\mu = n(2^{1/n} - 1)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Possible to exploit full computational power of processor without provisioning for slack</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* Unless the plug-in by Slomka et al. is added.
Sporadic tasks

If sporadic tasks were connected to interrupts, the execution time of other tasks would become very unpredictable.

- Introduction of a sporadic task server, periodically checking for ready sporadic tasks;
- Sporadic tasks are essentially turned into periodic tasks.
Dependent tasks

The problem of deciding whether or not a schedule exists for a set of dependent tasks and a given deadline is NP-complete in general [Garey/Johnson].

Strategies:

1. Add resources, so that scheduling becomes easier

2. Split problem into static and dynamic part so that only a minimum of decisions need to be taken at run-time.

3. Use scheduling algorithms from high-level synthesis
Summary

Periodic scheduling

- Rate monotonic scheduling
- EDF
- Dependent and sporadic tasks (briefly)