Handling Sporadic Tasks in Off-Line Scheduled Distributed Real Time Systems

Damir Isović & Gerhard Fohler
Department of Computer Engineering
Mälardalen University, Sweden

Presented by: Aseem Lalani
Outline

- Introduction
- Paper Contribution
- System Description & Task Model
- Slot Shifting Method for integration of Online & Offline Scheduling
- Offline Guarantee Acceptance Test
- Offline Feasibility Test for Sporadic Tasks
- Example
- Conclusion
Introduction

- Acquisition of temporal aspects of application is either difficult or impossible to gather due to high costs or unavailability.
- Knowledge of only partial information (such as minimum inter-arrival time between tasks) of the controlled environment.
- Online Systems provide no countermeasures for sporadic task sets which are rejected.
Paper Contribution

- Method for off-line feasibility test for sporadic tasks on top of off-line scheduled distributed periodic set.
- Ability to re-schedule or re-design upon test failure.
System Description

- System is distributed viz. consists of several processing and comm. nodes
- Discrete Time Model with task periods and deadlines defined in terms of slot-length.
Task Model

- Number of slots defined by LCM of involved periods.
- Periodic Task $T_p$ characterized by Max. Execution Time (MAXT), Period (P), and relative deadline (DI).
- Hard Aperiodic tasks characterized by arrival time (a), maximum execution time and relative deadline.
- No deadline constraints for soft aperiodic jobs.
Task Model (cont’d)

- Sporadic tasks arrive at random points in time with defined minimum inter-arrival times between two consecutive invocations.
- Arrival order pattern not known but Max. Frequency of arrival of sporadic tasks is known.
- Sporadic Task $T_s$ characterized by relative deadline, minimum inter-arrival time ($\lambda$) and Max. Execution Time.
- Additional On-line information available about sporadic tasks include arrival time of $k^{th}$ invocation is its arrival time and its absolute deadline.
Slot Shifting for Integrated off-line and on-line scheduling

- Efficient method to provide on-line guarantee of scheduling aperiodic tasks on top of a distributed schedule with task constraints.
- Re-claims unused resources from off-line schedule to schedule other feasible tasks.
- Off-line preparations include
  - Allocation of tasks to nodes, resolving precedence constraints by ordering task execution.
  - Creating schedule tables listing start and end times of task executions.
-> Creating disjoint intervals for each node with tasks having the same deadline constituting one interval.

-> Calculating spare capacity for interval $I_i$ as:

$$sc(I_i) = |I_i| - \sum_{T \in I_i} \max(T) - \min(sc(I_{i+1}), 0)$$
Guarantee Algorithm for Aperiodic Tasks

- At each slot, guarantee algorithm is performed on arriving aperiodic tasks.
- For each aperiodic task $T_A$, find
  - $A = \text{sc}(I_c)_t$: Spare remaining capacity of current interval.
  - $B = \text{Positive Spare capacities of full intervals between } t \text{ and } dl(T_A)$.
  - $C = \text{Min(sc of last interval, execution need of } T_A \text{ before its deadline in this interval)}$.
- $(A + B + C) > \text{MAXT}(T_A)$ guarantees acceptance of $T_A$
On-line Scheduling

- $sc(I_C) > 0 \implies$ Apply EDF to set of Ready Tasks.
- $sc(I_C) = 0 \implies$ Guaranteed task has to execute else task deadline violation will occur.
- Soft Aperiodic Tasks execute immediately when $sc(I_C) > 0$
- After each scheduling decision, update spare capacities of affected intervals.
Acceptance Test for Sporadic Tasks

- Feasible set is defined to schedule all tasks in the sporadic set such that no periodic task misses its deadline.

- The test includes:
  -> Creating an off-line schedule for periodic tasks analyzed for slot-shifting.
  -> Fitting sporadic tasks by investigating critical time slots.
  -> Re-design the system upon failure of test to manage the sporadic tasks.
More on Sporadic Tasks

- Sporadic tasks have been proven to behave like periodic tasks for worst case analysis when successive tasks arrive at minimum inter-arrival time.
- Guaranteeing this worst case load pattern at critical time slots guarantees acceptance of all sporadic tasks with greater inter-arrival times.

![Critical slot diagram](Image)

Figure 1. Example of a critical slot.

Critical slot is defined as the time slot when the execution of sporadic tasks can be delayed maximally.
Why Critical Slots are important to investigate?

- If the sporadic set can be fitted within the periodic set upon arrival at critical slot then it can be guaranteed to fit upon arrival at any other slot.

\[ \Delta = \text{Difference between spare capacities for sporadic task } T_S \text{ at critical slot } t_c \text{ and any other slot } t. \]

\[ \alpha = \text{Difference in spare capacity of the arrival caused by shifting arrival time } T_S \text{ from } t_c \text{ to } t. \]

\[ \beta = \text{Difference in spare capacity of the deadline interval caused by shifting the deadline of } T_S. \]

\[ \Delta = \alpha + \beta \]
Critical Slot Investigation

$t > t_c \Rightarrow T_S\text{ arrives after C.S.}$

$\alpha = 0$
$\beta \geq 0$
$\Delta = (\alpha + \beta) \geq 0$

$t < t_c \Rightarrow T_S\text{ arrives before C.S.}$

$\beta_{\text{worst}} = -\alpha$
$|\beta_{\text{opt}}| < \alpha$
$\Delta = (\alpha + \beta) \geq 0$

Contradiction!
Off-line Feasibility Test for Sporadic Tasks

1: Investigate every critical slot.
2: No slots reserved yet.
3: Guarantee every sporadic task $T_S$ in the set.
4: Guarantee every invocation $T^n_s$ of $T_S$.
5: Calculate $sc$ available for $T_S$ from its arrival until its deadline. It is equal to the sum of $sc$ for all full intervals between $I_{arrival}$ and the $I_{deadline}$ of $T^n_s$, increased by the remaining $sc$ of the $I_{deadline}$ available until $dl(T^n_s)$, decreased by the amount of $sc$ reserved for other, previously guaranteed sporadics that intersect with $T^n_s$.
6: If the available $sc$ is greater or equal to the maximum execution time of $T_S$, then
7: reserve slots needed for $T^n_s$ as close to its $dl$ as possible, and continue.
10: If not enough spare capacity, abort the guarantee algorithm and report that the guaranteeing failed.
Example

(a) Precedence Graph & Task Description

(b) Task Execution at nodes

(c) Critical Slots and Intervals

(d) Schedule Table
Example (cont’d)

\[ S = \{S_1(1; 5); S_2(3; 10)\} \]

Steps in Guarantee Algorithm

<table>
<thead>
<tr>
<th>( t_c )</th>
<th>( T_s )</th>
<th>Inv.</th>
<th>( \text{sc}_a )</th>
<th>( \geq \text{MAXT}(?) )</th>
<th>( R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>( S_1 )</td>
<td>1</td>
<td>1</td>
<td>( \geq 1 \Rightarrow \top )</td>
<td>{5}</td>
</tr>
<tr>
<td>&amp; 2</td>
<td>3</td>
<td>( \geq 1 \Rightarrow \top )</td>
<td>{5,11}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( S_2 )</td>
<td>1</td>
<td>2</td>
<td>( \geq 3 \Rightarrow \bot )</td>
<td>abort</td>
<td></td>
</tr>
</tbody>
</table>

Critical Slot 3
Example (cont’d)

Guaranteeing after Re-design

\[ S = \{S1(1; 5); S2(3; 10)\} \]
Example (cont’d)

1. \( \mathcal{R}(t) = \{\} \): There are no tasks ready to be executed, the CPU remains idle.

2. \( \mathcal{R}(t) \neq \{\} \land \exists T_A, T_A \) soft aperiodic:
   
   (a) \( sc(I)_t > 0 \land \exists T_S \in \mathcal{R}(t), T_S \) sporadic \( \Rightarrow \) execute \( T_S \).

   (b) \( sc(I)_t > 0 \land \neg \exists T_S \in \mathcal{R}(t) \Rightarrow \) execute \( T_A \).

   (c) \( sc(I)_t = 0 \): a periodic task from ready set has to be executed. Zero spare capacities indicate that adding further activities will result in a deadline violation of the guaranteed task set.

3. \( \mathcal{R}(t) \neq \{\} \land \neg \exists T_A, T_A \) soft aperiodic: The task of ready set with the shortest deadline is executed.

<table>
<thead>
<tr>
<th>t</th>
<th>( \mathcal{R}(t) )</th>
<th>case</th>
<th>exe.</th>
<th>sc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{( T_1, T_5 )}</td>
<td>3</td>
<td>( T_1 )</td>
<td>unchanged</td>
</tr>
<tr>
<td>1</td>
<td>{( T_1, T_5 )}</td>
<td>3</td>
<td>( T_1 )</td>
<td>unchanged</td>
</tr>
<tr>
<td>2</td>
<td>{( T_5, A_1 )}</td>
<td>2b</td>
<td>( A_1 )</td>
<td>( sc(I_0) ) decreased</td>
</tr>
<tr>
<td>3</td>
<td>{( T_5, S_1^1, S_2, A_1 )}</td>
<td>2a</td>
<td>( S_1^1 )</td>
<td>( sc(I_0) ) decreased</td>
</tr>
<tr>
<td>4</td>
<td>{( T_5, S_2, A_1 )}</td>
<td>2a</td>
<td>( S_2 )</td>
<td>( sc(I_0) ) decreased</td>
</tr>
<tr>
<td>5</td>
<td>{( T_5, S_2, A_1 )}</td>
<td>2a</td>
<td>( S_2 )</td>
<td>( sc(I_1) ) decreased</td>
</tr>
<tr>
<td>6</td>
<td>{( T_5, A_1 )}</td>
<td>2b</td>
<td>( A_1 )</td>
<td>( sc(I_1) ) decreased</td>
</tr>
<tr>
<td>7</td>
<td>{( T_5 )}</td>
<td>3</td>
<td>( T_5 )</td>
<td>unchanged</td>
</tr>
<tr>
<td>8</td>
<td>{( T_5, S_1^2 )}</td>
<td>2c</td>
<td>( T_5 )</td>
<td>unchanged</td>
</tr>
</tbody>
</table>

On-line Execution at Node 0 with Aperiodic task \( A_1 \) (2,2)
Conclusion

- Sporadic Tasks are guaranteed during design time allowing re-scheduling or re-design in case of failure.
- Efficient method since major part of preparation is off-line and on-line mechanisms are simple.
- Slot shifting algorithm allows reclaim of unused resources allowing high resource utilization.