Non-Work-Conserving Scheduling in Non-Preemptive Event-Driven Real-Time Systems

Nathan van Ofwegen

Mitra Nasri
Real-Time Systems

A systems correctness depends on

- Logical correctness of results
- Timing correctness of results
What is scheduling?

Important to schedule the correct task!

Non-preemptive scheduling
Why non-preemptive scheduling?

- No preemption support
  - Small microcontrollers

- Timing predictability
  - More predictably cache

- Lower overhead
  - No context switches
Non-preemptive scheduling is hard

- No known optimal scheduling policy
- No known optimal idle-time insertion policy

Try all possible combinations of tasks and idle times...
Non-preemptive scheduling anomalies

Non-Preemptive Rate Monotonic (NP-RM)

Schedule the next pending job with the smallest period

Non-work-conserving

Before scheduling, verify no deadline miss.
If so? Wait...
Existing non-work-conserving policies

- **Precautious-RM (PRM)** [1]
  - Lower success rate in meeting deadlines (schedulability)
  - Small overhead: PRM checks deadline miss for only 1 task.

- **Critical-Window EDF (CW-EDF)** [2]
  - Higher schedulability
  - Considerable overhead: CW-EDF checks deadline miss for all tasks.

---

Precautious-RM


Check if the next job can finish before the latest start time (LST) of the next job of task 1
Limitations of Precautious-RM

- PRM only checks the LST of task 1.

- S is the smallest LST

1. Sort non-pending future jobs by deadline
2. Obtain the latest start time of these jobs, called $S$
3. Check if the current job can finish before start time $S$
Further limitations

CW-EDF and PRM require a concrete knowledge of the arrival times of the future jobs.

- Do not support periodic tasks with release jitter.
- Do not support event-triggered tasks.
Problem statement

• Design a policy for time-triggered tasks which has:
  • Lower overhead than CW-EDF
  • Higher schedulability than PRM

• Design a policy for event-triggered tasks which has:
  • Less deadline misses in a soft real-time environment
  • Workloads can be characterized by an arrival curve
Types of timing constraints

- Hard real-time
  - Missing the deadline may cause catastrophic consequences

- Soft real-time
  - Missing the deadline causes performance degradation
What is an arrival curve?

• An arrival curve represents the **lower bound** and **upper bound** on the inter-arrival time between consecutive events in a system.

• Arrival curves are obtained from measurements and are widely used in the industry.
Why handling event-triggered tasks with arrival curves is not straightforward?

- We need a history of prior events
  - Costs memory
  - History must be updated

- How to decide based on history and the curve?
  - Larger history leads to more overhead

Until 111 there is no arrival, because of the minimal arrival time between 4 consecutive jobs.
A non-work-conserving scheduling policy for time-triggered tasks

- Lower run-time overhead
- Equal schedulability to CW-EDF

A non-work-conserving scheduling policy for event-triggered tasks

- Reduce deadline misses
Building a new non-work-conserving scheduling policy for time-triggered tasks

K-Precautious-EDF (KP-EDF)
The intuition

- Use CW-EDF as a baseline
- Reduce the number of tasks we look at for each idle-time decision of CW-EDF

How to identify critical tasks?

How many of those tasks?
Implementation: finding critical tasks

Ignore the task with the latest deadline

Add to critical task set: \{1,3,4\}

Repeat for every decision in the schedule
Possible that the critical task set will contain all tasks!
KP-EDF: Achievements

Reduced overhead

KP-EDF looks at $k$ amount of tasks, instead of all other tasks

Equal schedulability

KP-EDF has the same idle-time decisions as CW-EDF
Evaluation of KP-EDF

- Inspect influence of
  - Number of tasks $n$
  - Utilization $U$

- Random workload generation methods
  - Automotive benchmarks [Kramer, et al. 2015]
  - Log-uniform period generation [Emberson, et al. 2010]

- Overhead measured on Arduino Mega
Automotive: critical task set size

Periods: 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000

KP-EDF has the same overhead as PRM

Log-uniform periods: critical task set size

Periods:

U = 0.3  
U = 0.6  
U = 0.9

Number of tasks

Task set size

KP-EDF has low overhead for log-uniform task sets

KP-EDF Overhead

Using log uniform task sets with a utilization of $[0.50,0.95]$


Non-Work-Conserving Scheduling in Non-Preemptive Event-Driven Real-Time Systems

KP-EDF Overhead

Using log uniform task sets with a utilization of [0.50,0.95]

Significant lower overhead up to 60%


Contributions

A non-work-conserving scheduling policy for time-triggered tasks

- Lower run-time overhead
- Equal schedulability to CW-EDF

A non-work-conserving scheduling policy for event-triggered tasks

- Reduce deadline misses
Building a new non-work-conserving scheduling policy for event-triggered tasks

Arrival Curve-EDF (ArC-EDF)
The intuition

- Use CW-EDF as baseline
- Use the arrival curve to estimate the next arrival

Estimate based on last arrivals in the history

Use the best estimation as next arrival time
Implementation: estimating next arrival

History: 8, 12, 19

Calculate multiple estimates of the next arrival

Add minimal inter-arrival time to history

Use maximum: 21
Implementation: different inter-arrival times

Pick value between the minimum and maximum inter-arrival times

For two consecutive events: between $a$ and $d$
For three consecutive events: between $c$ and $f$
...

One set for the entire policy

ArC-EDF with chosen values is one policy
Achievements: ArC-EDF

Reducing deadline misses

Using the arrival curve to estimate next arrival
Evaluation of ArC-EDF

- Inspect impact of
  - Number of tasks \( n \)
  - Utilization \( U \)
  - History size
  - Release jitter factor

- Based on log uniform task sets
  - Add random (uniform) release jitter to time-triggered tasks
ArC-EDF: Impact of number of tasks

ArC-EDF inserts many idle-times when using the minimum inter-arrival time

ArC-EDF: Impact of utilization

ArC-EDF has less deadline misses when using the average inter-arrival time

ArC-EDF: Impact of history size

Size 1

Size 5

Size 10

History has no impact on deadline misses
ArC-EDF: Impact of release jitter factor

More uncertainty leads to false idle-time insertions

n = 8
U = 0.8
History = 1

Summary

- Designed a time-triggered policy (KP-EDF)
  - Lower overhead (up to 60%)
  - Equal schedulability to CW-EDF

- Designed an event-triggered policy (ArC-EDF)
  - Inserts many idle-times
  - History has no effect
Future work

• KP-EDF
  • Calculate the critical tasks

• ArC-EDF
  • Arrival curves and history might have an effect for non-uniform distributed inter-arrivals
Thank you

- **KP-EDF**: Efficient policy for time-triggered tasks

- **ArC-EDF**: Arrival curve has no effect on deadline misses and inserts too many idle-times
ArC-EDF: Why history has no impact

More history only introduces less likely scenario’s

History of 1 is the best estimation in 96%
Logarithmic: generation

Randomly pick $n$ periods
Randomly divide $U$ between the $n$ tasks

$U: [0.1, \ldots, 0.9]$

$n: [1, \ldots, 20]$
## Automotive: task set generation

<table>
<thead>
<tr>
<th>Period (ms)</th>
<th>Avg execution time (us)</th>
<th>Distribution (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.00</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4.20</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>11.01</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>10.1</td>
<td>25</td>
</tr>
<tr>
<td>20</td>
<td>8.75</td>
<td>25</td>
</tr>
<tr>
<td>50</td>
<td>17.56</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>10.53</td>
<td>20</td>
</tr>
<tr>
<td>200</td>
<td>2.56</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>0.43</td>
<td>4</td>
</tr>
</tbody>
</table>

Keep picking and merging runnables from the table until \( U \) is reached.
ArC-EDF: Work-conserving