FIFO WITH OFFSETS

HIGH SCHEDULABILITY WITH LOW OVERHEADS

RTAS’18
April 18, 2018

Mitra Nasri
Rob Davis
Björn Brandenburg
FIFO SCHEDULING

First-In-First-Out (FIFO) scheduling

- extremely simple
- very low overheads

- very low schedulability

ideal for:
- IoT-class devices
- deeply embedded systems
- hardware implementations

meeting deadlines?
FIFO SCHEDULING

First-In-First-Out (FIFO) scheduling

- extremely simple
- very low overheads

ideal for:
- IoT-class devices
- deeply embedded systems
- hardware implementations

meeting deadlines?

VERY HIGH!

THIS PAPER

FIFO can actually achieve excellent schedulability!

[periodic non-preemptive tasks on a uniprocessor]
INTUITION
FIFO WITH OFFSETS: HIGH SCHEDULABILITY WITH LOW OVERHEADS

THE PROBLEM WITH PLAIN FIFO SCHEDULING

**FIFO** schedule of 3 periodic tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_3$</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

Missed
Plain FIFO is oblivious to deadlines and priorities

$\tau_3$ comes first $\rightarrow$ deadline miss

**FIFO schedule of 3 periodic tasks:**

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_3$</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>
The problem with plain FIFO scheduling

FIFO schedule of 3 periodic tasks:

\[
\begin{array}{c|c|c}
\text{Task} & \text{WCET} & \text{Period} \\
\tau_1 & 3 & 10 \\
\tau_2 & 6 & 12 \\
\tau_3 & 8 & 24 \\
\end{array}
\]

In fact, any work-conserving policy (EDF, RM, …) must schedule \( \tau_3 \) here \( \rightarrow \) deadline miss.
NON-WORK-CONSERVING SCHEDULING

\[ \text{critical-window EDF: Nasri & Fohler, 2016} \]

\textbf{CW-EDF} schedule of the same 3 periodic tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_3 )</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>( \tau_1 )</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

Idle time
CW-EDF considers *future job arrivals* in the “critical window” and *postpones* $\tau_3$ until later.

**CW-EDF** schedule of the same 3 periodic tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_3$</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>
NON-WORK-CONSERVING SCHEDULING

CW-EDF schedule of the same 3 periodic tasks:

LIMITATION

CW-EDF incurs much higher runtime overheads than simple work-conserving policies.

ATMega2560 @ 16 MHz: 9.2× higher than RM!
INTUITION: FIFO + "JUST THE RIGHT" OFFSETS

FIFO schedule + offset for $\tau_3$:

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_3$</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>
INTUITION: FIFO + “JUST THE RIGHT” OFFSETS

FIFO schedule + offset for $\tau_3$:

Move $\tau_3$ “out of the way” by introducing (or adjusting) a release offset.

FIFO schedule becomes identical to CW-EDF schedule!
INTUITION: FIFO + "JUST THE RIGHT" OFFSETS

FIFO schedule + offset for $\tau_3$:

CW-EDF schedule is identical:

Idle time

[Altmeyer, Sundharam, & Navet, 2016]
THIS PAPER
OFFSET TUNING ALGORITHM
Given a set of $n$ periodic non-preemptive tasks, find, for each job of each task, a release offset such that

(A) the resulting FIFO schedule is feasible, and

(B) the number of offsets per task is minimized.

**Challenges**
- space of possible offsets is large and unstructured
- even ignoring (B), solving “just” (A) is very difficult

**Altmeyer et al.**
- randomize offsets + test
- not systematic
- scalability limitations

KEY INSIGHT

Given a set of $n$ periodic non-preemptive tasks, find, for each job of each task, a release offset such that

(A) the resulting **FIFO schedule is feasible**, and

(B) the **number of offsets** per task is minimized.

Solving (A) is very difficult... so we don’t!

OFFSET TUNING

*Infer offsets* from a given **feasible reference schedule**, while greedily working towards (B).
OFFSET TUNING – OVERVIEW

- Generate feasible schedule
- Reference schedule
- Offset tuning algorithm
- Offset compression
  - Offset vectors
  - Compact offset table

Simple FIFO scheduler + job release offsets

Or...
- CW-EDF [Nasri & Fohler, 2016]
- Or ILP/SAT solving
- Or bespoke planning heuristics
- Or...

Given task set
SCHEDULE EQUIVALENCY

A schedule $S_1$ is equivalent to $S_2$ if

(i) they schedule the \textit{same jobs},

(ii) in the \textit{same order}, and

(iii) \textit{jobs start no later} in $S_1$ than in $S_2$.

Non-preemptive execution

$\rightarrow$ jobs also complete no later in $S_1$ than in $S_2$

Offset Tuning

$\rightarrow$ ensures FIFO schedule is equivalent to reference schedule
POI: POTENTIAL OFFSETS INTERVAL

POI of a job: range of release offsets that guarantee schedule equivalency.

FIFO schedule + offset for $\tau_3$:
POI: POTENTIAL OFFSETS INTERVAL

POI of a job: range of release offsets that guarantee schedule equivalency.

FIFO schedule + offset for $\tau_3$:

POI: any release time of $\tau_3$ in (12, 19] will yield an equivalent schedule.
OFFSET PARTITION

Consecutive jobs of a task form an offset partition if they have *mutually intersecting POIs*.

→ can be assigned a single offset

→ offset partitioning not necessarily unique
OFFSET TUNING ALGORITHM (SIMPLIFIED)

for each task $\tau_i$ in deadline-monotonic order:

**greedily** create **offset partitions** for $\tau_i$

**assuming** jobs of larger-deadline tasks
are released as in reference schedule
for each task $\tau_i$ in deadline-monotonic order:

- **greedily** create *offset partitions* for $\tau_i$

  *assuming jobs of larger-deadline tasks are released as in reference schedule*

Release times of not-yet-processed jobs still unknown $\Rightarrow$ **speculate**.

*Mis-speculation increases the number of offset partitions,*

*but *does not* cause the algorithm to fail.*
PROPERTIES OF OFFSET TUNING

REFERENCE SCHEDULE EQUIVALENCY

In the resulting FIFO schedule, no job completes later than in the original reference schedule.

PER-TASK MINIMAL OFFSET PARTITIONS

The greedy offset partitioning strategy yields a minimal number of offset partitions (for a given task).

NON-MINIMAL OFFSET PARTITIONS FOR ENTIRE TASK SET

Deadline-monotonic processing order does not guarantee overall minimal number of offset partitions (but works well empirically).
What if we want just a *single offset* per task?

- no extra memory required
- compatibility with existing systems

**FST: First-Start-Time Heuristic**

- pick start time of first job in reference schedule

**FOP: First-Offset-Partition Heuristic**

- pick offset from first offset partition of the task
EVALUATION
EVALUATION QUESTIONS

Q1: Does FIFO + Offset Tuning still have low runtime overheads?

Q2: Does FIFO + Offset Tuning (FIFO-OT) significantly improve schedulability relative to EDF/RM?

Q3: How many offsets are assigned?

Q4: How much memory is needed?
PROTOTYPE PLATFORM

Arduino Mega 2560
ATMega2560 microcontroller
16 MHz CPU
256 KiB Flash
8 KiB SRAM (no cache)

gcc: -Os

http://people.mpi-sws.org/~bbb/papers/details/rtas18
### EVALUATED SCHEDULERS

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP-RM</td>
<td>plain non-preemptive rate-monotonic scheduling</td>
</tr>
<tr>
<td>NP-EDF</td>
<td>plain non-preemptive EDF</td>
</tr>
<tr>
<td>CW-EDF</td>
<td>Critical Window EDF [Nasri &amp; Fohler, 2016]</td>
</tr>
<tr>
<td>TD</td>
<td>Table-driven (a.k.a. static or time-triggered)</td>
</tr>
<tr>
<td>OE</td>
<td>Offline Equivalence [Nasri &amp; Brandenburg, 2017]</td>
</tr>
<tr>
<td>FIFO-OT</td>
<td>FIFO + Offset Tuning [this paper]</td>
</tr>
</tbody>
</table>
Q1: RUNTIME OVERHEADS

- Max ▼ Min — Avg

<table>
<thead>
<tr>
<th>Overhead (microseconds)</th>
<th>3 tasks</th>
<th>6 tasks</th>
<th>9 tasks</th>
<th>12 tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>CW-EDF</td>
<td>76</td>
<td>72</td>
<td>104</td>
<td>136</td>
</tr>
<tr>
<td>NP-EDF</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>44</td>
</tr>
<tr>
<td>NP-RM</td>
<td>36</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>OE</td>
<td>44</td>
<td>52</td>
<td>60</td>
<td>68</td>
</tr>
<tr>
<td>FIFO-OT</td>
<td>44</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>TD</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

3 tasks      6 tasks      9 tasks      12 tasks
LOW RUNTIME OVERHEADS

FIFO-OT is much cheaper than CW-EDF and roughly similar to NP-RM and OE.
WORKLOADS

based on

Kramer, Ziegenbein, and Hamann, "Real world automotive benchmark for free," WATERS 2015

Periods

→ non-uniformly in \{1, 2, 5, 10, 20, 50, 100, 200, 1000\} milliseconds

Runnable BCETs and WCETs

→ randomly generated based on statistics provided by Kramer et al.

Runnable Packing

→ Runnables aggregated into tasks until random utilization threshold reached

→ utilization threshold ensures feasibility under non-preemptive scheduling
Q2: SCHEDULABILITY GAINS

The graph shows the schedulability ratio as a function of utilization for different scheduling algorithms:

- **NP-RM**: High schedulability ratio even at high utilization.
- **Plain FIFO**: Lower schedulability ratio compared to NP-RM.
- **FIFO + FST**: Moderate schedulability gains over plain FIFO.
- **FIFO + FOP**: Significant schedulability gains compared to FIFO + FST.
- **FIFO + Offset Tuning**: Highest schedulability gains among all algorithms, maintaining high schedulability even at high utilizations.
As expected, plain FIFO exhibits very low schedulability.
Assigning even a **single offset** per task can substantially increase schedulability!
FIFO-OT achieves *much higher schedulability*, thanks to CW-EDF reference schedule.
Most tasks require only few offset partitions.
Across the hyper-period, offsets values repeat cyclicly.

→ Opportunity to store offsets efficiently (compression).
Numbers of unique offsets per task set

Up to 25× reduction in the number of offset values that must be stored.

Across the hyper-period, offsets values repeat cyclicly.

→ Opportunity to store offsets efficiently (compression).
MEMORY USAGE

- offset tuning
- offline equivalence
- table driven

percentage of task sets vs. required memory (B)
[non-linear scale]
Both OE and FIFO-OT require much less memory than table-driven scheduling. 

*dozens to hundreds of bytes vs. 10KiB-20KiB*
For a fraction of task sets, OE requires slightly less memory (< 100 bytes difference)…
\[...\text{but FIFO-OT can support over 90\% of task sets with } \leq 250 \text{ bytes of offset data.}\]
IMPLEMENTATION FOOTPRINT

- **code size**
- **global data (for 12 tasks)**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory Used (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP-FP</td>
<td>312</td>
</tr>
<tr>
<td>TD</td>
<td>414</td>
</tr>
<tr>
<td>NP-EDF</td>
<td>490</td>
</tr>
<tr>
<td>FIFO-OT</td>
<td>824</td>
</tr>
<tr>
<td>CW-EDF</td>
<td>928</td>
</tr>
<tr>
<td>OE</td>
<td>1,004</td>
</tr>
</tbody>
</table>

**NP-FP**
- Code Size: 48
- Global Data: 10

**TD**
- Code Size: 10
- Global Data: 108

**NP-EDF**
- Code Size: 156
- Global Data: 215

**FIFO-OT**
- Code Size: 229
- Global Data: 229

**CW-EDF**
- Code Size: 215
- Global Data: 215

**OE**
- Code Size: 1,004
- Global Data: 1,004
About 150 bytes smaller footprint than OE (RAM + code).

<table>
<thead>
<tr>
<th></th>
<th>code size</th>
<th>global data (for 12 tasks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP-FP</td>
<td>312</td>
<td>48</td>
</tr>
<tr>
<td>TD</td>
<td>414</td>
<td>10</td>
</tr>
<tr>
<td>NP-EDF</td>
<td>490</td>
<td>108</td>
</tr>
<tr>
<td>FIFO-OT</td>
<td>824</td>
<td>229</td>
</tr>
<tr>
<td>CW-EDF</td>
<td>928</td>
<td>156</td>
</tr>
<tr>
<td>OE</td>
<td>1,004</td>
<td>215</td>
</tr>
</tbody>
</table>

memory used (byte)

- **NP-FP**: About 150 bytes smaller footprint than OE (RAM + code).
IMPLEMENTATION FOOTPRINT

- code size
- global data (for 12 tasks)

About 650 bytes more than most simple implementation (RAM + code).

<table>
<thead>
<tr>
<th></th>
<th>NP-FP</th>
<th>TD</th>
<th>NP-EDF</th>
<th>FIFO-OT</th>
<th>CW-EDF</th>
<th>OE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory used (byte)</td>
<td>312</td>
<td>414</td>
<td>490</td>
<td>824</td>
<td>928</td>
<td>1,004</td>
</tr>
<tr>
<td>Code size</td>
<td>48</td>
<td>10</td>
<td>108</td>
<td>229</td>
<td>156</td>
<td>215</td>
</tr>
<tr>
<td>Global data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory used for the FIFO-OT implementation is significantly higher than for other schedulers, with about 650 bytes more than the most simple implementation. This includes both RAM and code size.
CONCLUSION
FIFO SCHEDULING

First-In-First-Out (FIFO) scheduling

- extremely simple
- very low overheads

- ideal for:
  - IoT-class devices
  - deeply embedded systems
  - hardware implementations

very high schedulability
meeting deadlines?

THIS PAPER

FIFO can actually achieve excellent schedulability!
(periodic non-preemptive tasks on a uniprocessor)

OFFSET TUNING – OVERVIEW

generate feasible schedule given task set
reference schedule
offset tuning algorithm
offset compression
offset vectors
offset table

offline online

simple FIFO scheduler + job release offsets

PROPERTIES OF OFFSET TUNING

REFERENCE SCHEDULE EQUIVALENCY

In the resulting FIFO schedule, no job completes later than in the original reference schedule.

PER-TASK MINIMAL OFFSET PARTITIONS

The greedy offset partitioning strategy yields a minimal number of offset partitions (for a given task).

NON-MINIMAL OFFSET PARTITIONS FOR ENTIRE TASK SET

Deadline-monotonic processing order does not guarantee overall minimal number of offset partitions (but works well empirically).

Q2: SCHEDULABILITY GAINS

FIFO-OT achieves much higher schedulability, thanks to CW-EDF reference schedule.
APPENDIX
CAN OFFSET TUNING BE APPLIED TO EDF OR FIXED-PRIORITY SCHEDULING?

→ yes in principle, but no equivalence guarantee

**FIFO** schedule + *offset for τ₃* :

**RM** schedule + *offset for τ₃* :

Missed