FIFO WITH OFFSETS

HIGH SCHEDULABILITY WITH LOW OVERHEADS

RTAS'18
April 13, 2018

Mitra Nasri
Rob Davis
Björn Brandenburg
FIFO SCHEDULING

First-In-First-Out (FIFO) scheduling

- extremely simple
- very low overheads

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

very low schedulability

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 low schedulability

HIGH!

 THIS PAPER

FIFO can actually achieve excellent schedulability!

[periodic non-preemptive tasks on a uniprocessor]
INTUITION
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>$\tau_1$</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>8</td>
<td>60</td>
</tr>
</tbody>
</table>

Missed
THE PROBLEM WITH PLAIN FIFO SCHEDULING

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:

In fact, *any work-conserving policy* (EDF, RM, ...) must schedule $\tau_3$ here $\rightarrow$ deadline miss.
**Non-Work-Conserving Scheduling**

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>
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>$\tau_1$</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>8</td>
<td>60</td>
</tr>
</tbody>
</table>


CW-EDF considers future job arrivals in the “critical window” and postpones $\tau_3$ until later.
NON-WORK-CONSERVING SCHEDULING

[critical-window EDF: Nasri & Fohler, 2016]

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** $\tau_3$:

[Altmeyer, Sundharam, & Navet, 2016]

**CW-EDF** schedule is identical:
THIS PAPER
OFFSET TUNING ALGORITHM
PROBLEM STATEMENT

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

given task set

reference schedule

offset tuning algorithm

offset compression

offset vectors

compact offset table

online

offline

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

simple FIFO scheduler + job release offsets
SCHEDULE EQUIVALENCY

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

(i) they schedule the same jobs,

(ii) in the same order, and

(iii) jobs start no later in $S_1$ than in $S_2$.

Non-preemptive execution

→ jobs also complete no later in $S_1$ than in $S_2$

Offset Tuning

→ 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** $\tau_3$:

- $\tau_3$:
  - Offset for $\tau_3$: 8

- $\tau_2$:
  - Release offsets: 6, 12, 19

- $\tau_1$:
  - Release offsets: 3, 10, 20, 30, 40, 50, 60

- Missed:
  - FIFO schedule + offset for $\tau_3$: 19
**POI: POTENTIAL OFFSETS INTERVAL**

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

**FIFO** schedule + offset for $\tau_3$: 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
Need to start somewhere…

**shorter relative deadline = fewer options**

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`

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

Overhead (microseconds)

Max  Min  Avg

3 tasks
CW-EDF: 76  48  36
NP-EDF: 156  72  44
NP-RM: 156  72  44
OE: 156  72  44
FIFO-OT: 156  72  44
TD: 156  72  44

6 tasks
CW-EDF: 104  64  60
NP-EDF: 308  104  104
NP-RM: 308  104  104
OE: 308  104  104
FIFO-OT: 308  104  104
TD: 308  104  104

9 tasks
CW-EDF: 404  136  136
NP-EDF: 404  136  136
NP-RM: 404  136  136
OE: 404  136  136
FIFO-OT: 404  136  136
TD: 404  136  136

12 tasks
CW-EDF: 76  68  68
NP-EDF: 76  68  68
NP-RM: 76  68  68
OE: 76  68  68
FIFO-OT: 76  68  68
TD: 76  68  68

3, 6, 9, 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

![Graph showing schedulability gains with different FIFO configurations. The graph plots the schedulability ratio against utilization. Different lines represent various FIFO configurations: NP-RM, plain FIFO, FIFO + FST, FIFO + FOP, and FIFO + offset tuning. The graph demonstrates the benefits of using FIFO with offsets in terms of high schedulability with low overheads.](image-url)
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.
Q3: NUMBERS OF OFFSETS PER TASK

Most tasks require only few offset partitions.
Across the hyper-period, offsets values repeat cyclicly.

→ Opportunity to store offsets efficiently (compression).
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

percentage of task sets

required memory (B)

[non-linear scale]

offset tuning
offline equivalence
table driven

<= 5 <= 20 <= 100 <= 200 <= 300 <= 400 <= 500 <= 600 <= 700 <= 800 <= 900 <= 1000 <= 1100 <= 1250 <= 1500 <= 3000 <= 7500 <= 12500 <= 20000

memory usage

percentage of task sets
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)…
...but FIFO-OT can support over 90% of task sets with ≤ 250 bytes of offset data.
IMPLEMENTATION FOOTPRINT

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

<table>
<thead>
<tr>
<th>Scheduler</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 (for 12 tasks)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>Method</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>

Memory used (byte)
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 low schedulability

meeting deadlines?

THIS PAPER

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

OFFSET TUNING – OVERVIEW

generate feasible schedule

reference schedule

offset tuning algorithm

offset compression

offset vectors

compact 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 $\tau_3$:

---

**RM** schedule + *offset* for $\tau_3$: