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

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 high schedulability

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**
Plain FIFO is oblivious to deadlines and priorities. 

τ₃ comes first → deadline miss

**THE PROBLEM WITH PLAIN FIFO SCHEDULING**: The FIFO schedule of 3 periodic tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>τ₂</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>τ₃</td>
<td>8</td>
<td>60</td>
</tr>
</tbody>
</table>

Diagram:
- τ₁: 10, 3
- τ₂: 12, 24
- τ₃: 8

FIFO schedule with offsets (o₃ = 19)

NP-FP schedule with rate-monotonic priorities and offsets (o₃ = 19)

Plain FIFO is oblivious to deadlines and priorities.
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

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

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:

**Task** | **WCET** | **Period**
---|---|---
$\tau_3$ | 8 | 60
$\tau_2$ | 6 | 12
$\tau_1$ | 3 | 10
**NON-WORK-CONSERVING SCHEDULING**

*critical-window EDF: Nasri & Fohler, 2016*

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

![Diagram of CW-EDF schedule with task durations and idle time](image)

**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 τ₃:**

---

<table>
<thead>
<tr>
<th>Task</th>
<th>WCET</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₃</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>τ₂</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>τ₁</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:

[Altmeyer, Sundharam, & Navet, 2016]
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

- **given task set**
  - generate feasible schedule
  - reference schedule
  - offset tuning algorithm
  - offset compression
    - offset vectors
    - compact offset table
- **online**
  - 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 τ₃:**

Diagram showing the POI of a job, with τ₃, τ₂, and τ₁ tasks, and release offsets for each task.
**POI: POTENTIAL OFFSETS INTERVAL**

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

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

**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 \textit{mutually intersecting POIs}.

\[ \rightarrow \text{can be assigned a single offset} \]

\[ \rightarrow \text{offset partitioning not necessarily unique} \]
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).
SINGLE-OFFSET HEURISTICS

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

NP-RM plain non-preemptive rate-monotonic scheduling

NP-EDF plain non-preemptive EDF

CW-EDF Critical Window EDF [Nasri & Fohler, 2016]

TD Table-driven (a.k.a. static or time-triggered) scheduling

OE Offline Equivalence [Nasri & Brandenburg, 2017]

FIFO-OT FIFO + Offset Tuning [this paper]
Q1: Runtime Overheads

<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>32</td>
<td>36</td>
<td>44</td>
</tr>
<tr>
<td>NP-RM</td>
<td>36</td>
<td>52</td>
<td>64</td>
<td>76</td>
</tr>
<tr>
<td>OE</td>
<td>44</td>
<td>32</td>
<td>32</td>
<td>68</td>
</tr>
<tr>
<td>FIFO-OT</td>
<td>44</td>
<td>56</td>
<td>60</td>
<td>32</td>
</tr>
<tr>
<td>TD</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
</tbody>
</table>

- Max
- Min
- Avg

- CW-EDF
- NP-EDF
- NP-RM
- OE
- FIFO-OT
- TD

- 3 tasks
- 6 tasks
- 9 tasks
- 12 tasks

Max overhead: 404 microseconds
Min overhead: 76 microseconds
Average overhead: 104 microseconds

Tasks: 3, 6, 9, 12
LOW RUNTIME OVERHEADS

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

<table>
<thead>
<tr>
<th></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>48</td>
<td>72</td>
<td>32</td>
<td>44</td>
</tr>
<tr>
<td>NP-EDF</td>
<td>36</td>
<td>32</td>
<td>36</td>
<td>44</td>
</tr>
<tr>
<td>NP-RM</td>
<td>44</td>
<td>56</td>
<td>64</td>
<td>76</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>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>TD</td>
<td></td>
<td></td>
<td></td>
<td>404</td>
</tr>
</tbody>
</table>

Max • Min • Avg

0 50 100 150 200 250 300 350 400 450

overhead (microseconds)

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 across different utilization levels. The graph compares various scheduling mechanisms, including NP-RM, plain FIFO, FIFO + FST, FIFO + FOP, and FIFO + offset tuning. The x-axis represents utilization, ranging from 0.1 to 0.9, and the y-axis represents schedulability ratio, ranging from 0 to 1. The graph illustrates the performance improvements achieved by incorporating offsets in FIFO scheduling.]
As expected, plain FIFO exhibits very low schedulability.
Assigning even a single offset per task can substantially increase schedulability!
**Q2: SCHEDULABILITY GAINS**

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

percentage of task sets

required memory (B)
[non-linear scale]

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

- offset tuning
- offline equivalence
- table driven

MEMORY USAGE

percentage of task sets

required memory (B)
[non-linear scale]

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

- offset tuning
- offline equivalence
- table driven
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>Process</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, TD, NP-EDF, FIFO-OT, CW-EDF, OE
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).

NP-FP: 312 bytes
TD: 414 bytes
NP-EDF: 490 bytes
FIFO-OT: 824 bytes
CW-EDF: 928 bytes
OE: 1,004 bytes

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

given task set

generate feasible schedule

reference schedule

offset tuning algorithm

offset compression

offset vectors

offset table

online

offline

compact offset table

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.
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 $\tau_3$: