Hard Real-Time Scheduling with Cache-Related Preemption Delays

Guillaume PHAVORIN  Pascal RICHARD

October 15th, 2015

Seminar: http://labex-digicosme.fr/GT+OVSTR
Outline

Part I - Survey

- Background
  - Caching problems
  - Cache-related scheduling problem

- Cache-Related Scheduling Problems: State-of-The-Art
  - Main approaches
  - Classification and open issues

Part II - Several contributions

- CRPD-aware scheduling
  - Adding CRPD in the task model and analysis
    (optimality, sustainability, computational complexity)
  - Offline method and numerical results

- Cache-aware Scheduling
  - Adding cache state information in the task model
  - Computational complexity analysis
Context

Two major resources, that must be fully utilized

- computation: processors are fast
- memory: main memory is very slow and caches are very small

- Cache-Related Preemption Delays ($\text{CRPD}$) can represent up to 44% of the overall Worst-Case Execution Time ($\text{WCET}$) [1]–[4].

How to define efficient scheduling techniques for these resources...

- Where is the bottleneck (processors, caches, bus contention,...combination of them)?
- What is the impact of caches on scheduling problems?
Which Cache metrics for Real-Time Systems?

Hard Real-Time Systems

- **CRPD** bounds and Enhanced Schedulability Analysis (Jan Reineke’s talk).
- Using **CRPD** and cache information to take scheduling decisions (this talk)

Soft-real Time Systems (not covered hereafter)

- Working Set Size: amount of cache lines accessed by a task (memory footprint)
- Stack Distance: number of different cache-lines accessed between two consecutive accesses to the same reference
  - ...(see Hennessy and Patterson [5] for additional metrics)
  - ...(see Calendrino and Anderson [6] for soft real-time syst.)
Caching Problems

- Current processors have cache memories

- Caches are resources with their own management unit (Replacement algorithm)

- \( \text{cost(cache miss)} \gg \text{cost(cache hit)} \)

- Optimized for a single stream of sequential block requests
Cache replacement policy

Clairvoyant optimal algorithm [7]
(Belady’s 1966 algorithm) Evict the block that will be used the furthest in the future

Online algorithms (competitive analysis [8])
- Adversary: optimal sequence for a size-$k$ cache causes $m$ misses.
- No (deterministic) on-line algorithm for a size-$k$ cache can have a competitive ratio better than $k$ (i.e., $k \times m$ cache evictions)

Basic observations
- Caches are mainly designed to reduce intra-task cache misses
- Periodic real-time tasks generate numerous inter-task misses (due to preemptions)
**Cache-Related Preemption Delays (CRPD)**

**CRPD**

**Additional reloads** because of cache evictions due to **preempting jobs**
Cache-Related Preemption Delays (CRPD)

Additional reloads because of cache evictions due to preemitting jobs

$\tau_i$ access to A

$\tau_j$ access to B

MISS
**Cache-Related Preemption Delays (CRPD)**

**CRPD**

**Additional reloads** because of cache evictions due to *preempting jobs*

![Diagram of cache and access](image)

- **Cache:**
  - Block A
  - Block B

- **Access to A:**
  - $\tau_i$

---

Hard Real-Time Scheduling with Cache-Related Preemption Delays
Cache-Related Preemption Delays (CRPD)

Additional reloads because of cache evictions due to preemption jobs

$\tau_i$: access to A
Cache-Related Preemption Delays (CRPD)

Additional reloads because of cache evictions due to preemting jobs

\( \tau_i \)

access to A

\( A \)

\( \tau_j \)

access to B

\( A \)

\( B \)
Cache-Related Preemption Delays (CRPD)

Additional reloads because of cache evictions due to preempting jobs

\[ \tau_i \] access to A

\[ \tau_j \] access to B

MISS

A

cache
Cache-Related Preemption Delays (CRPD)

**CRPD**

Additional reloads because of cache evictions due to preemption jobs

\[ \tau_i \] access to A

\[ \tau_j \] access to B

![Diagram showing cache evictions and reloads](attachment:image.png)
Cache-Related Preemption Delays (CRPD)

**CRPD**

*Additional reloads because of cache evictions due to preemtpting jobs*

- $\tau_j$: access to B
- $\tau_i$: access to A
- MISS

Block Reload Time (BRT)
Cache-Related Preemption Delays (CRPD)

Additional reloads because of cache evictions due to preemption jobs
Cache-Related Preemption Delays (CRPD)

CRPD

Additional reloads because of cache evictions due to preemining jobs

↗ System utilization (Up to 44% [1]–[4])

\[ \tau_i \]

\[ \tau_j \]

access to A

Block Reload Time (BRT)

access to B

cache
Cache-Related Preemption Delays (CRPD)

Additional reloads because of cache evictions due to preemining jobs

System utilization (Up to 44% [1]–[4])

predictability?
Cache-Related Preemption Delays (CRPD)

**Additional reloads** because of cache evictions due to preempting jobs

- System utilization (Up to 44% [1]–[4])
- Predictability?
- Schedulability?
Scheduling Theory: Basic Task models

Periodic (Sporadic) task

- Every task $\tau_i$ releases an infinite set of jobs $(\tau_{ij}, j = 1..\infty)$
- Every job as the same WCET $C_i$
- Jobs are periodically released every $T_i$ units of time (minimum interarrival time in the sporadic case)
- Jobs are subjected to a deadline $D_i$ units of time after their releases.

Deadlines

- Implicit: $D_i = T_i$
- Constrained: $D_i \leq T_i$
- Arbitrary: otherwise.
Outline

1 Background
   • Cache-related problems
   • Cache-related scheduling problem

2 State-of-the-Art
   • Main approaches
   • Classification and open issues

3 CRPD-aware scheduling problem
   • Problem statement and classical analyses
   • Optimal offline scheduling method and numerical results

4 Cache-aware scheduling problem
   • Problem statement
   • Complexity analysis

5 Concluding remarks and perspectives
State-of-the-art (Tentative)

Main Approaches [9]

- Bounding $\text{CRPD}$ and introducing it in schedulability tests
  - $\text{WCET}$ community
- Cache management: partitioning/locking cache lines
  - Platform community
- Controlling preemptions points: static and dynamic approaches
  - Scheduling Community
Bounding the CRPD

- Platform features (CPU, cache...)
- Task code
- Task periods and deadlines
- Task WCET
- Timing analysis
- Scheduler
- Schedulability analysis
- Yes/No
Bounding the CRPD

Platform features (CPU, cache...) → Task code → Task periods and deadlines → Task CRPD → Task WCET → Static cache analysis

Timing analysis → Task CRPD

Scheduler → Schedulability analysis → Yes/No

Lee et al. (1997, [10])

Preempted task → Useful Cache Blocks (UCBs)
Bounding the CRPD

- Task code
- Task periods and deadlines
- Task WCET
- Static cache analysis
- Schedulability analysis
- Scheduler
- Timing analysis
- Task CRPD
- Platform features (CPU, cache...)

- preempted task
  - Useful Cache Blocks (UCBs)
- preemtioning task
  - Evicting Cache Blocks (ECBs)

Busquets-Mataix et al. (1996, [11])
Bounding the CRPD

Platform features (CPU, cache...)

Timing analysis

Task code

Task periods and deadlines

Task CRPD

Scheduler

Task WCET

Static cache analysis

Schedulability analysis

Static cache analysis

Yes/No

➡️ preempted task
  ← Useful Cache Blocks (UCBs)

➡️ preemting task
  ← Evicting Cache Blocks (ECBs)

➡️ combined approaches
  ← both tasks

Altmeyer et al. (2012, [12])
Bounding the CRPD

- preempted task
  - Useful Cache Blocks (UCBs)

- preemining task
  - Evicting Cache Blocks (ECBs)

- combined approaches
  - both tasks

- improvements:
  - Definitely-Cached UCBs

- Altmeyer et al. (2009, [13])

Platform features (CPU, cache...)

Task code

Task periods and deadlines

Scheduler

Task WCET

Static cache analysis

Schedulability analysis

Yes/No
Bounding the CRPD

Platform features (CPU, cache...)

Task code

Task periods and deadlines

Timing analysis

Task CRPD

Task WCET

Scheduler

Schedulability analysis

CRPD added to the WCET

Yes/No

\[ \text{WCET}_{w/o \text{ preemption}} + n \times \text{CRPD} \]

\[ \leftarrow n? \]

\textbf{Altmeyer et al. (2011, [14])}
Bounding the CRPD

- Platform features (CPU, cache...)
- Task code
- Task periods and deadlines
- Task WCET

**Fixed-Priority (FP):**

\[ R_i = C_i + \sum \left\lceil \frac{R_i}{T_j} \right\rceil \cdot (C_j + \gamma_{i,j}) \]

**CRPD in sched. analysis**

*Busquets-Mataix et al. (1996, [11])*
Bounding the CRPD

- **Platform features (CPU, cache...)**
- **Task code**
- **Task periods and deadlines**
- **Task WCET**
- **Scheduler**

- **Fixed-Priority (FP):**
  \[ R_i = C_i + \sum \left\lceil \frac{R_i}{T_j} \right\rceil \cdot (C_j + \gamma_{i,j}) \]

- **EDF scheduling:**
  Similar to Fixed-Priority

**Lunniss et al., (2013,[15])**
Cache management

Platform features (CPU, cache...)

Task code

Task periods and deadlines

Timing analysis

Task WCET

Task CRPD

Scheduler

Schedulability analysis

Yes/No
Cache management

Platform features (CPU, cache...):

- Task code
- Task periods and deadlines

Task code

- Timing analysis
- Task CRPD

Task WCET

Scheduler

Schedulability analysis

Yes/No

- Bui et al. (2008,[16])

Cache partitioning

- $\tau_1$
- $\tau_2$
- $\tau_3$

main memory

cache
Cache management

- cache partitioning
- cache locking → cache content fixed ⇒ predictability

Ding et al. (2013,[17])

Platform features (CPU, cache...)
Task code
Task periods and deadlines
Task WCET
Task CRPD
Scheduler
Schedulability analysis

Yes/No
Cache management

- cache partitioning

- cache locking
  - cache content fixed
  - predictability

- memory layout
  - code positioning
    - WCET
  - task positioning
    - CRPD

- Lunniss et al (2012,[18])
Scheduling (Controlling preemptions)

- Platform features (CPU, cache...)
- Task code
- Task periods and deadlines
- Task WCET
- Task CRPD
- Schedulability analysis
- Scheduler
- Yes/No

State-of-the-Art
Main approaches
Scheduling (Controlling preemptions)

- Preemption Thresholds
  preemption possible only if: priority(preempting task) > threshold(preempted task)

Platform features (CPU, cache...)
Task code
Task periods and deadlines
Timing analysis
Task CRPD
Scheduler
Task WCET
Schedulability analysis
Yes/No

Bril et al. (2014,[19])
Scheduling (Controlling preemptions)

- **Preemption Thresholds**
  - preemption possible only if: $\text{priority}(\text{preempting task}) > \text{threshold}(\text{preempted task})$

- **Deferred Preemptions**
  - preemption postponed as much as possible

---

- **Platform features** (CPU, cache...)
- **Task code**
- **Task periods and deadlines**
- **Task WCET**
- **Scheduler**
- **Timing analysis**
- **Task CRPD**
- **Schedulability analysis**

---

*Bertogna et al. (2010,[20])*
Scheduling (Controlling preemptions)

Platform features (CPU, cache...)

Task code

Task periods and deadlines

Task CRPD

Scheduler

Schedulability analysis

Task WCET

Timing analysis

Platform features (CPU, cache...)

Scheduling (Controlling preemptions)

Preemption Thresholds

- preemption possible only if: priority(preempting task) > threshold(preempted task)

Deferred Preemptions

- preemption postponed as much as possible

Fixed Preemptive Points

- Offline Stage: preemptions allowed only at precomputed program points, to minimize CRPD;
- Online stage: standard online priority scheduling.

Bertogna et al. (2011,[21])

Hard Real-Time Scheduling with Cache-Related Preemption Delays
## State-of-the-art (Tentative)

<table>
<thead>
<tr>
<th><strong>category</strong></th>
<th><strong>techniques</strong></th>
<th><strong>references</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory Management</strong></td>
<td>cache partitioning</td>
<td>[22]–[25]</td>
</tr>
<tr>
<td></td>
<td>fully-partitioning</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hybrid-partitioning</td>
<td>[16], [26]</td>
</tr>
<tr>
<td></td>
<td>cache locking</td>
<td></td>
</tr>
<tr>
<td></td>
<td>full locking</td>
<td>[27]–[33]</td>
</tr>
<tr>
<td></td>
<td>partial locking</td>
<td>[17], [34]</td>
</tr>
<tr>
<td></td>
<td>partitioning + locking</td>
<td>[35]</td>
</tr>
<tr>
<td></td>
<td>memory layout</td>
<td>[18], [36]–[42]</td>
</tr>
<tr>
<td><strong>WCET</strong></td>
<td>WCET integrating cache analysis</td>
<td>[43]–[48]</td>
</tr>
<tr>
<td></td>
<td>CRPD in WCET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>preemitting task</td>
<td>[49], [50]</td>
</tr>
<tr>
<td></td>
<td>preempted task</td>
<td>[13], [14], [51]</td>
</tr>
<tr>
<td></td>
<td>both tasks</td>
<td>[52]–[54]</td>
</tr>
</tbody>
</table>
## State-of-the-art (Tentative (Cont’))

<table>
<thead>
<tr>
<th>category</th>
<th>techniques</th>
<th>references</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched.</td>
<td><strong>CRPD in Sched. Analysis</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Preempting task</td>
<td>[11], [55]</td>
</tr>
<tr>
<td></td>
<td>Preempted task</td>
<td>[56]</td>
</tr>
<tr>
<td></td>
<td>both tasks</td>
<td>[12], [15], [57], [58]</td>
</tr>
<tr>
<td>Preemption</td>
<td>Floating Non Preemptive Region</td>
<td>[59], [60]</td>
</tr>
<tr>
<td>Control</td>
<td><strong>CRPD as a function</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Preemption Thresholds</td>
<td>[19], [61]</td>
</tr>
<tr>
<td></td>
<td>Opt+<strong>CRPD in Sched. Analysis</strong></td>
<td></td>
</tr>
</tbody>
</table>
State-of-the-art (Tentative (Cont'))

<table>
<thead>
<tr>
<th>category</th>
<th>techniques</th>
<th>references</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched. (Cont’)</td>
<td>Preemption Control (Cont’)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deferred Preemption</td>
<td>opt. + CRPD in Sched. Analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[4], [21], [62]–[65]</td>
</tr>
</tbody>
</table>

**State-of-the-art scheduling techniques**

- Integrating CRPD into existing scheduling techniques (i.e., FP, EDF, and their variants are considered).
- Results mainly for direct-mapped caches
Open issues (from scheduling point of view)

Approved!!

Timing and Schedulability Analysis must take into account CRPD.

Which granularity of the task model for improving scheduling decisions?

- fine-grained: exploiting cache state information?
- coarse-grained: worst-case CRPD as the maximum preemption delay for a task set, for a task, for a segment within a task,…?

Scheduling algorithms

- Extend and mix known techniques?
- Need of new approaches?
Cache impact on optimally taking scheduling decisions

Objective: define the most basic task models to cover:
- the largest set of scheduling problems
- all cache architectures (direct-mapped, Set-Associative and Full-Associative)
  - scheduling with cache-related preemption delays
    - crpd-aware scheduling problem
  - scheduling with cache state information
    - Cache-aware scheduling problem

Current collaborations
- Claire Maiza (Verimag, Grenoble)
- Joël Goossens (ULB, Brussels)
Outline

1 Background
   • Cache-related problems
   • Cache-related scheduling problem

2 State-of-the-Art
   • Main approaches
   • Classification and open issues

3 CRPD-aware scheduling problem
   • Problem statement and classical analyses
   • Optimal offline scheduling method and numerical results

4 Cache-aware scheduling problem
   • Problem statement
   • Complexity analysis

5 Concluding remarks and perspectives
CRPD scheduling problem statement

Scheduling decisions exploit cache-related preemption delays
→ **minimize the total overhead** (i.e., sum of CRPDs).

Task model: \( \tau_i(C_i, D_i, T_i, \gamma) \)
- \( C_i \): WCET without CRPD
  \( \rightarrow \) \( \tau_i \) executed fully non preemptively
- \( \gamma \): worst-case CRPD for one preemption
  \( \rightarrow \) the same for all program points and all tasks

Presented results [66], [67]
- Optimality and Complexity Analyses
- Sustainability Analysis (scheduling anomalies)
- Competitive Analysis (i.e., online v.s. offline schedulers)
- Optimal offline scheduling method and numerical results
Optimality Analysis

Example \((\tau_i(C_i, D_i, T_i, \gamma)): \tau_1(1, 3, 3, 0.6), \tau_2(7, 12, 12, 0.6))\)
Optimality Analysis

Example \((\tau_i(C_i, D_i, T_i, \gamma))\): \(\tau_1(1, 3, 3, 0.6), \tau_2(7, 12, 12, 0.6)\)

- Fixed-Task/Fixed-Job Priority Scheduling:

\[
\begin{align*}
\tau_1 & \quad \tau_2 \\
0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 & \quad 6 & \quad 7 & \quad 8 & \quad 9 & \quad 10 & \quad 11 & \quad 12 \\
\tau_1 & \quad \tau_2 & \quad \gamma & \quad \gamma & \quad \gamma & \quad \gamma & \quad \gamma & \quad \gamma & \quad \gamma & \quad \gamma & \quad \gamma & \quad \gamma & \quad \gamma
\end{align*}
\]

\(\rightarrow\) not schedulable
Optimality Analysis

Example $\tau_i(C_i, D_i, T_i, \gamma)$: $\tau_1(1, 3, 3, 0.6)$, $\tau_2(7, 12, 12, 0.6)$

- Fixed-Task/Fixed-Job Priority Scheduling:

- CRPD-aware scheduling:

  $\rightarrow$ not schedulable

  $\rightarrow$ schedulable
Optimality Analysis

Example \((\tau_i(C_i, D_i, T_i, \gamma))\): \(\tau_1(1, 3, 3, 0.6), \tau_2(7, 12, 12, 0.6)\)

- Fixed-Task/Fixed-Job Priority Scheduling:
  - → not schedulable

- CRPD-aware scheduling:
  - → schedulable

⇒ Fixed-Task and Fixed-Job Priority schedulers → not optimal.
Optimality Analysis (Cont’)

Finite set of synchronous tasks $\tau_i(C_i, D_i, T_i, \gamma)$,

Is there a uniprocessor preemptive schedule meeting the deadlines?
Optimality Analysis (Cont’)

Finite set of synchronous tasks $\tau_i(C_i, D_i, T_i, \gamma)$,

$\Rightarrow$ Is there a uniprocessor preemptive schedule meeting the deadlines?

$\Rightarrow$ **NP-hard in the strong sense.**

*Proof* [67]: transformation from the 3-Partition decision problem
Optimality Analysis (Cont’)

Proof (Sketch)

3-Partition [68]

- Instance: a set $A$ of $3m$ elements, a bound $B \in \mathbb{N}$, and a size $s_j \in \mathbb{N}$ for each $j = 1..3m$ such that $B/4 < s_j < B/2$ and $\sum_{j=1..3m} s_j = mB$.
- Question: Can $A$ be partitioned into $m$ disjoint sets $A_1, A_2, ..., A_m$ such that, for $1 \leq i \leq m$, $\sum_{j \in A_i} s_j = B$ (each $A_i$ must therefore contain exactly three elements from $A$)?

Transformation: To every 3-Partition instance

- $3m$ tasks $\tau_1, \ldots, \tau_{3m}$ with the parameters: $C_i = s_i, D_i = T_i = m(B + 1), 1 \leq i \leq 3m$.
- Task $\tau_{3m+1}$ with: $C_{3m+1} = D_{3m+1} = 1$ and $T_i = (B + 1)$
Optimality Analysis (Cont’)

Proof sketch (Cont’): In the task set defined from a 3-Partition instance:

- the workload is 100% without preemption
- In every feasible schedule, the pattern follows:

\[
\begin{align*}
\tau_{3m+1} & \quad B & \tau_{3m+1} & \quad B & \tau_{3m+1} & \quad \cdots & \tau_{3m+1} & \quad B \\
0 & 1 & B+1 & 2(B+1) & (m-1)(B+1) & m(B+1)
\end{align*}
\]

Property

A schedule is feasible if, and only if, there is no preemption (one preemption leads to a workload $> 100\%$).
Optimality Analysis (Cont’)

- Set of jobs with 2 distinct releases and deadlines and $\gamma = 1$

$\Rightarrow$ **NP-hard in the weak sense.**

*Proof* [69]: transformation from the 2-Partition decision problem
Optimality Analysis (Cont’)

- Set of jobs with 2 distinct releases and deadlines and $\gamma = 1$

$\Rightarrow$ NP-hard in the weak sense.

Proof [69]: transformation from the 2-Partition decision problem

- EDF still optimal for finite set of jobs with either:
  - $r_i = 0$
  - $d_i = d$
  - $C_i = 1$
  - similarly ordered release times and deadlines:
    $r_i \leq r_j \Rightarrow d_i \leq d_j$

$\Rightarrow$ since no job is preempted using EDF
Sustainability analysis

Sustainability definition
A scheduling policy is sustainable if any systems deemed schedulable remain schedulable if:

- a WCET is decreased
- a period is increased
- a relative deadline is increased
- a preemption delay is decreased

Known results for uniprocessor without CRPD [70]

- EDF is sustainable with respect to: WCET, period, deadlines
- Fixed-Task Priority ($RM, DM$) is sustainable w.r.t.: WCET, deadlines
EDF, RM and DM non Sustainability w.r.t. WCET

- 4 tasks (τᵢ(Cᵢ, Dᵢ, Tᵢ, γ)): τ₁(1, 4, 4, 0.6), τ₂(3, 12, 12, 0.6), τ₃(3, 12, 12, 0.6) and τ₄(3, 12, 12, 0.6)
- EDF, RM and DM use the same job priority assignment (task index as tie breaker).

→ schedule with \( C_2 = 3 \)
EDF, RM and DM non Sustainability w.r.t. WCET

4 tasks \((\tau_i(C_i, D_i, T_i, \gamma))\): \(\tau_1(1, 4, 4, 0.6)\), \(\tau_2(3, 12, 12, 0.6)\), \(\tau_3(3, 12, 12, 0.6)\) and \(\tau_4(3, 12, 12, 0.6)\)

- EDF, RM and DM use the same job priority assignment (task index as tie breaker).

\[ \rightarrow \text{ schedule with } C_2 = 3 \]  \[ \rightarrow \text{ schedule with } C_2 = 2 \]
**EDF, RM and DM non Sustainability w.r.t. Preemption delays**

- 4 tasks \( (\tau_i(C_i, D_i, T_i, \gamma)) \): \( \tau_1(1, 4, 4, 1) \), \( \tau_2(3, 12, 12, 1) \), \( \tau_3(3, 12, 12, 1) \) and \( \tau_4(3, 12, 12, 1) \)

- EDF, RM and DM use the same job priority assignment (task index as tie breaker).

\[ 
\begin{array}{c}
\tau_1 & \uparrow & J_{11} \\
\tau_2 & \uparrow & J_{21} \\
\tau_3 & \uparrow & J_{31} \\
\tau_4 & \uparrow & J_{41} \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12
\end{array} 
\]

\[ \rightarrow \text{schedule with } \gamma = 1 \]
EDF, RM and DM non Sustainability w.r.t. Preemption delays

- 4 tasks ($\tau_i(C_i, D_i, T_i, \gamma)$): $\tau_1(1, 4, 4, 1)$, $\tau_2(3, 12, 12, 1)$, $\tau_3(3, 12, 12, 1)$ and $\tau_4(3, 12, 12, 1)$
- EDF, RM and DM use the same job priority assignment (task index as tie breaker).

$\rightarrow$ schedule with $\gamma = 1$  $\rightarrow$ schedule with $\gamma_{31} \leq 1$
EDF non Sustainability w.r.t. relative deadlines

- 3 tasks \( (\tau_i(C_i, D_i, T_i, \gamma)) \): \( \tau_1(1, 3, 4, 1) \), \( \tau_2(2, 4, 6, 1) \), \( \tau_3(3, 6, 12, 1) \)
- EDF (task index as tie breaker).

\[ \rightarrow \text{EDF schedule with } D_3 = 6 \]
EDF non Sustainability w.r.t. relative deadlines

- 3 tasks \((\tau_i(C_i, D_i, T_i, \gamma))\): \(\tau_1(1, 3, 4, 1)\), \(\tau_2(2, 4, 6, 1)\), \(\tau_3(3, 6, 12, 1)\)
- EDF (task index as tie breaker).

\[ \rightarrow \text{EDF schedule with } D_3 = 6 \]  \[ \rightarrow \text{EDF schedule with } D_3 = 11 \]
EDF non Sustainability w.r.t. periods

➢ 3 tasks \( \tau_i(C_i, D_i, T_i, \gamma) \): \( \tau_1(2, 3, 4, 1) \), \( \tau_2(2, 5, 6, 1) \), \( \tau_3(2, 12, 12, 1) \)
➢ Shifting one release time (sporadic tasks)

→ EDF schedule with \( T_1 = 4 \)
EDF non Sustainability w.r.t. periods

» 3 tasks \(\tau_i(C_i, D_i, T_i, \gamma)\): \(\tau_1(2, 3, 4, 1)\), \(\tau_2(2, 5, 6, 1)\), \(\tau_3(2, 12, 12, 1)\)

» Shifting one release time (sporadic tasks)

\[\rightarrow\] EDF schedule with \(T_1 = 4\)  \[\rightarrow\] EDF schedule with \(T_1 = 5\)
EDF non Sustainability w.r.t. periods

- 3 tasks \((\tau_i(C_i, D_i, T_i, \gamma))\): \(\tau_1(2, 3, 4, 1), \tau_2(2, 5, 6, 1), \tau_3(2, 12, 12, 1)\)

- periodic tasks

\[ \rightarrow \text{EDF schedule with } T_2 = 6 \]
EDF non Sustainability w.r.t. periods

3 tasks \((\tau_i(C_i, D_i, T_i, \gamma))\): \(\tau_1(2, 3, 4, 1)\), \(\tau_2(2, 5, 6, 1)\), \(\tau_3(2, 12, 12, 1)\)

periodic tasks

\(\rightarrow\) EDF schedule with \(T_2 = 6\) \hspace{1cm} \(\rightarrow\) EDF schedule with \(T_2 = 7\)
FP and critical instant worst-case scenario

- 2 tasks $(\tau_i(C_i, D_i, T_i, \gamma))$: $\tau_1(1, 3, 3, 0.5)$, $\tau_2(5, 9, 9, 0.5)$
- sporadic tasks scheduled by RM
- Similar examples in [47], [71]

$$\rightarrow \text{wcrt}(\tau_2) = 9$$
**FP and critical instant worst-case scenario**

- 2 tasks \((\tau_i(C_i, D_i, T_i, \gamma))\): \(\tau_1(1, 3, 3, 0.5)\), \(\tau_2(5, 9, 9, 0.5)\)
- sporadic tasks scheduled by **RM**
- Similar examples in [47], [71]

\[
\rightarrow \text{wcrt}(\tau_2) = 9 \\
\rightarrow \text{wcrt}(\tau_2) = 9.5
\]
Competitive analysis

Online scheduling model

- Set of jobs released over time
- at each job release, all its parameters are known

Result: Optimal online scheduling is impossible

Job release times need to be a priori known to define an optimal online scheduler (i.e., clairvoyant).
Competitive analysis (Cont’)

Proof sketch

Optimal offline scheduler (the adversary) generates jobs so that any online scheduler cannot define a feasible schedule whereas the adversary can.

Adversary strategy:

1. Generate two jobs $J_1(0,5,12,1)$ and $J_2(4,5,6,1)$
2. At time 4
   - Case 1 the online scheduler continues to execute $\tau_1$
   - Case 2 the online scheduler preempts $\tau_1$ to execute $\tau_2$
3. According to the case, the adversary defines a new job $\tau_3$. 
Proof sketch (Cont’)

**Case 1.** The online scheduler continues to execute Job $\tau_1$ at time 4. Adversary generates a job $J_3(9,1,1,1)$.

→ Online algorithm
Proof sketch (Cont’)

**Case 1.** The online scheduler continues to execute Job $\tau_1$ at time 4. Adversary generates a job $J_3(9,1,1,1)$.

→ Online algorithm  

→ Adversary’s feasible schedule
Proof sketch (Cont’)

**Case 2.** the online scheduler preempts $\tau_1$ to execute $\tau_3$ at time 4 $\tau_2$. Adversary generates a job $J_3(10,1,1,1)$.

→ Online algorithm
Case 2. the online scheduler preempts $\tau_1$ to execute $\tau_3$ at time 4 $\tau_2$.
Adversary generates a job $J_3(10,1,1,1)$.

→ Online algorithm

→ Adversary’s feasible schedule
Offline scheduling: MILP formulation

Mixed Integer Linear Program (MILP)

Define an offline schedule to:

- Minimize the total workload
- or equivalently, minimize the total preemption delay (since \( WCET \) contributes as a constant in the objective function)

Constraints:

- each job is executed for its \( WCET \)
- each job is executed between its release time and its deadline
- at most one job is executed at any time instant
- explicit preemption delay
MILP: schedule construction

Schedule Construction

- The schedule is defined as a finite set of slices $S_j$, separated by release dates/deadlines (No job release inside a slice).
- In every slice, job-pieces and their related preemption delays must fit in the slice interval.

Main property to define a feasible schedule

Every job is executed at most once in every slice with no pmtn.

MILP Variables to define the schedule in each slice $S_j$

- $t_{i,j} \in \mathbb{R}$: starting time of job-piece $\tau_i$ in $S_j$
- $p_{ij} \in \mathbb{R}$: execution time of job-piece $\tau_i$ in $S_j$
- $\Delta_{ij} \in \{0, 1\}$: job-piece $\tau_i$ has to pay a pmtn. delay in $S_j$. 
MILP: schedule construction (Cont’)

Rewriting simplification

- Execution time modification: $C'_i = C_i - s_i$ (i.e., the first job-piece pays a fictive pmtn delay at release)
- Simpler formulation of pmtn. penalty constraints in the MILP

Simple example with two periodic tasks

Five jobs $J_i(r_i, C_i, s_i, D_i)$ generated within [0,12)

- Task 1: $J_1(0, 1, 0.2, 3)$, $J_2(3, 2, 1, 0.2, 3)$, $J_3(6, 1, 0.2, 3)$, $J_4(9, 1, 0.2, 3)$
- Task 2: $J_5(0, 7, 0.5, 12)$

It can be easily checked that EDF generates 3 pmtn and misses a deadline.
MILP: schedule construction (Cont’)

- 4 Slices: $S_1 = [0, 3), S_2 = [3, 6), S_3 = [6, 9), S_4 = [9, 12)$
- 8 job-pieces (rows in the following table)

MILP results:

<table>
<thead>
<tr>
<th>job-pieces job</th>
<th>slice</th>
<th>Variables $t_{ij}$</th>
<th>$C'_{ij}$</th>
<th>$\Delta_{ij}$</th>
<th>pmtn $s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1</td>
<td>0</td>
<td>0.8</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>$J_5$</td>
<td>1</td>
<td>1</td>
<td>1.5</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>$J_5$</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>$J_2$</td>
<td>2</td>
<td>5</td>
<td>0.8</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>$J_3$</td>
<td>3</td>
<td>6</td>
<td>0.8</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>$J_5$</td>
<td>3</td>
<td>7.5</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>$J_5$</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>$J_4$</td>
<td>4</td>
<td>11</td>
<td>0.8</td>
<td>1</td>
<td>0.2</td>
</tr>
</tbody>
</table>

→ Optimal schedule
Experiments

Synthetic task sets

- \((C_i, T_i)\) UUnifast to generate utilization factors
- \(s_i\) randomly generated between 0 and the Maximum Pmtn Delay Factor (PDF, percentage of the WCET \(C_i\))
- Limited to 200 jobs over the hyperperiod (to limit CPLEX running time)

Monitored Algorithms

- EDF based algorithms
  - EDF: arbitrary tie breaker
  - LP-EDF: tie breaker avoid unnecessary preemptions
  - Cache-related schedulability analysis (Lunniss et al. RTAS’2013 [15])
- OPT: MILP solved by CPLEX 12.6.1
Results: Total Utilization

Experiment parameters

Maximum Preemption Delay Factor (PDF) 20%.

![Graph showing total processor utilization and schedulable tasksets]
Results: Weighted Schedulability

Experiment parameters

Weighted Schedulability as a function of the Maximum PDF.

\[ Q = \{ u | u = k \cdot 0.1, k \in [1, 10] \} \]

\[ W_\ell(PDF) = \frac{\sum_{\forall U \in Q} U \cdot S_\ell(U, PDF)}{\sum_{\forall U \in Q} U} \]

with \( S_\ell(U, PDF) \) binary sched. result
Results: Number of preemptions

Experiment parameters

Average number of preemptions per job as function of U (PDF=20%).

![Graph showing average number of preemptions per job as function of total processor utilization.](image)
Results: Total preemption delays

Experiment parameters

Total preemption delays as a function of maximum PDF.
Outline

1 Background
   - Cache-related problems
   - Cache-related scheduling problem

2 State-of-the-Art
   - Main approaches
   - Classification and open issues

3 CRPD-aware scheduling problem
   - Problem statement and classical analyses
   - Optimal offline scheduling method and numerical results

4 Cache-aware scheduling problem
   - Problem statement
   - Complexity analysis

5 Concluding remarks and perspectives
Cache-aware scheduling problem

Scheduling with cache state information

Cache assumptions

- consists in a single cache line,
- miss cost = $BRT$ (Block Reload Time), hit=0
- block references: string of letters

Simple Job model

synchronous Job $i$: $J_i(C_i, D, S_i)$:

- $C_i$: WCET without cache miss,
- $D$: a common relative deadline for all jobs
- $S_i$: sequence of memory blocks used during the job execution
  - One memory block used per time unit
  - no if-then-else structure
Example: $J_1(5, 13, cbabd), J_2(4, 13, eba f), \text{BRT} = 0.5$

\[ S_1 = c \rightarrow b \rightarrow a \rightarrow b \rightarrow d, \quad S_2 = e \rightarrow b \rightarrow a \rightarrow f \]
Example: \( J_1(5, 13, \text{cbabd}), J_2(4, 13, \text{ebaf}) \), BRT = 0.5

\[ S_1 = c \rightarrow b \rightarrow a \rightarrow b \rightarrow d, \quad S_2 = e \rightarrow b \rightarrow a \rightarrow f \]

Fixed-Job Priority Scheduling

\((\text{prio}(J_1) > \text{prio}(J_2))\):

→ not schedulable
Example: \( J_1(5, 13, cbabd), J_2(4, 13, ebaf), \text{BRT} = 0.5 \)

\[ S_1 = c \rightarrow b \rightarrow a \rightarrow b \rightarrow d, \quad S_2 = e \rightarrow b \rightarrow a \rightarrow f \]

- **Fixed-Job Priority Scheduling**
  \((\text{prio}(J_1) > \text{prio}(J_2)):\)

  \[ J_1 \quad c \quad b \quad a \quad b \quad d, \quad J_2 \quad e \quad b \quad a \quad f \]

  → not schedulable

- **Cache-aware scheduling:**

  \[ J_1 \quad c \quad b \quad a \quad b \quad d, \quad J_2 \quad e \quad b \quad a \quad f \]

  → schedulable
Example: \(J_1(5, 13, cbabd), J_2(4, 13, eba f), \text{BRT} = 0.5\)

\[S_1 = c \rightarrow b \rightarrow a \rightarrow b \rightarrow d, \quad S_2 = e \rightarrow b \rightarrow a \rightarrow f\]

- **Fixed-Job Priority Scheduling**  
  \((\text{prio}(J_1) > \text{prio}(J_2)):\)

- **Cache-aware scheduling:**

  \[\Rightarrow \text{Fixed-Task and Fixed-Job Priority schedulers} \rightarrow \text{not optimal}.\]
Complexity result

Finite set of $n$ jobs $J_i(C_i, D, S_i)$ with a common deadline $D$:

→ a uniprocessor preemptive schedule meeting the overall deadline $D$ for every job $J_i$?
Complexity result

Finite set of $n$ jobs $J_i(C_i, D, S_i)$ with a common deadline $D$

⇒ a uniprocessor preemptive schedule meeting the overall deadline $D$ for every job $J_i$?

⇒ NP-hard in the

**strong sense.**

*Proof (see [67]):* Transformation from the Shortest Common Supersequence problem known to be strongly NP-Complete [68]*
Concluding remarks and perspectives

Outline

1. Background
   - Cache-related problems
   - Cache-related scheduling problem

2. State-of-the-Art
   - Main approaches
   - Classification and open issues

3. CRPD-aware scheduling problem
   - Problem statement and classical analyses
   - Optimal offline scheduling method and numerical results

4. Cache-aware scheduling problem
   - Problem statement
   - Complexity analysis

5. Concluding remarks and perspectives
Conclusion

- **Real-Time Scheduling with Cache-Related Pmtn Delays**

- **CRPD-aware scheduling problem**
  - Scheduling anomalies for standard scheduling policies (i.e., FP, EDF)
  - **NP-hard** in the strong sense
    - no pseudo-polynomial optimal scheduling algorithm
  - No optimal online scheduling policies (set of jobs)

- **Cache-aware scheduling problem**
  - RM, EDF are not optimal
  - **NP-hard** in the strong sense
    - no pseudo-polynomial optimal scheduling algorithm
Perspectives

Continue to study basic task models
- to understand underlying theoretical problems
- to evaluate the performance of a given technique
- to delimit practical solutions for real-world problems

Scheduling under cache constraints
- Uniprocessor: evaluate the loss of schedulability of online schedulers
- Multiprocessors: timing issues are quite tricky
Circular dependencies must be taken into account

- Timing issues: CRPD computation under constraints (cache partitioning, Code positioning, enforced scheduling isolation,...)
- Compilers: code/task positioning to avoid bottlenecks in caches
- Cache management: partitioning/locking techniques
- Schedulers: taking into account all previously mentioned constraints

Models and Metrics

- Shared task models to fully exploit them at design stages (timing, scheduling analysis, scheduler)
- Common metrics to measure the relative contributions of mixed techniques
Thank you!
Questions?
References


References (cont.)


References (cont.)


References (cont.)


References (cont.)


References (cont.)


