Integrating Cache Related Pre-emption Delay Analysis into EDF Scheduling

Will Lunniss\textsuperscript{1} Sebastian Altmeyer\textsuperscript{2} Claire Maiza\textsuperscript{3} Robert I. Davis\textsuperscript{1}

\textsuperscript{1}Real-Time Systems Research Group, University of York, UK
{wl510, rob.davis}@york.ac.uk

\textsuperscript{2}Department of Computer Science, Saarland University, Germany
altmeyer@cs.uni-sb.de

\textsuperscript{3}Verimag, Grenoble INP, Grenoble, France
Claire.maiza@imag.fr
Outline

• EDF Scheduling
• CRPD
• Integrating CRPD analysis into EDF
• Comparison with existing approach
• Improved CRPD analysis
• Case study
• Synthetic taskset evaluation
• Conclusions
Earliest Deadline First (EDF)

• It is dynamic scheduling algorithm
• Schedules the job of the task with the earliest absolute deadline first
• Proven to be optimal by Dertouzos on a single core processor
Determining which job should run

\[ \tau_1 \]
\[ \tau_2 \]
\[ \tau_3 \]
Determining which job should run

- If two jobs have the same absolute deadline
  - We assume that the job with the lowest task index is chosen
  - E.g. $\tau_2$ pre-empts $\tau_3$ in the above example
Determining which job should run

• If two jobs have the same absolute deadline
  – Ensures that two tasks cannot pre-empt each other
  – Ensures that after a pre-emption, the task that was pre-empted last is resumed first
  – E.g. $\tau_2$ is resumed at $t = 7$
Determining which job should run

• If two jobs have the same absolute deadline
  – Ensures that two tasks cannot pre-empt each other
  – Ensures that after a pre-emption, the task that was pre-empted last is resumed first
  – E.g $\tau_2$ is resumed at $t = 7$, rather than $\tau_3$
Determining which job should run

- Also applies for jobs with the same relative deadline and release time
  - E.g. $\tau_2$ is resumed at $t = 3$, rather than $\tau_3$ starting
Determining which job should run

• Also applies for jobs with the same relative deadline and release time
  – E.g. $\tau_2$ is resumed at $t = 3$, rather than $\tau_3$ starting
Determining which job should run

• Also applies for jobs with the same relative deadline and release time
  – E.g. $\tau_2$ is resumed at $t = 3$, rather than $\tau_3$ starting
Schedulability test

• If all tasks have implicit deadlines ($D_i = T_i$), schedulability test is
Schedulability test

• If all tasks have implicit deadlines \((D_i=T_i)\), schedulability test is

\[ U \leq 1 \]
Schedulability test

• If all tasks have implicit deadlines ($D_i = T_i$), schedulability test is

\[ U \leq 1 \]

• If $D_i \neq T_i$ then the test is still necessary, but is no longer sufficient
Schedulability test

• If all tasks have implicit deadlines \((D_i = T_i)\), schedulability test is

\[ U \leq 1 \]

• If \(D_i \neq T_i\) then the test is still necessary, but is no longer sufficient

• Need to do another test
Processor demand bound function

\[ h(t) = \sum_{i=1}^{n} (C_i - T_{dt}) \leq 0 \]
Processor demand bound function

\[ h(t) = \sum_{i=1} \]  

Sum over each task
Processor demand bound function

\[ h(t) = \sum_{i=1}^{n} \max \left\{ 0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right\} \]

Sum over each task

the number of jobs a task has which are released and have their deadlines in the interval \( t \)
Processor demand bound function

\[ h(t) = \sum_{i=1} \max \left\{ 0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right\} C_i \]

- Sum over each task
- the number of jobs a task has which are released and have their deadlines in the interval \( t \)
- multiplied by the tasks’ execution time
**Processor demand bound function**

\[ h(t) = \sum_{i=1}^{n} \max\left\{ 0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor \right\} C_i \]

- Sum over each task
- the number of jobs a task has which are released and have their deadlines in the interval \( t \)
- multiplied by the tasks’ execution time

Schedulability test

• A taskset is schedulable iff $h(t) \leq t$ for all values of $t$
  – The execution time requirement must be less than or equal to the available time
Schedulability test

• A taskset is schedulable iff $h(t) \leq t$ for all values of $t$
  – The execution time requirement must be less than or equal to the available time

• $h(t)$ can only change when $t$ is equal to an absolute deadline

• Bound the maximum value of $t$, $L$, using either
  – *Hyper-period*: Least common multiple of task periods
  – Synchronous busy period
Schedulability test

• There are still a large number of values for $t$ that need to be checked
Schedulability test

• There are still a large number of values for $t$ that need to be checked
• Can be reduced by using the Quick convergence Processor-demand Analysis (QPA) algorithm by Zhang and Burns
  – Starts with a value of $t$ close to $L$
  – Iterates back towards 0 checking a significantly smaller number of values
Schedulability test

• There are still a large number of values for $t$ that need to be checked

• Can be reduced by using the Quick convergence Processor-demand Analysis (QPA) algorithm by Zhang and Burns
  – Starts with a value of $t$ close to $L$
  – Iterates back towards 0 checking a significantly smaller number of values

Pre-emption and Cache
Related Pre-empt Delays (CRPD)
Pre-emptions and *Cache Related Pre-empt Delays (CRPD)*

- Pre-empting task can evict blocks belonging to the pre-empted task
- CRPD are introduced when the pre-empted task has to reload some of those evicted cache blocks after resuming
Cache block categorisation

• *Evicting Cache Blocks* (ECBs)
  – Loaded into cache and can therefore evict other blocks
Cache block categorisation

- *Evicting Cache Blocks* (ECBs)
  - Loaded into cache and can therefore evict other blocks

- *Useful Cache Blocks* (UCBs)
  - Reused once they have been loaded into cache before potentially being evict by the task
  - If evicted by another task, they may have to be reloaded which introduces CRPD
  - UCBs are always ECBs
Cache block categorisation

• Example block classification

ECBs  UCBs
Cache block categorisation

• Example block classification

• Instructions inside loops are often UCBs as they get reused
CRPD analysis

• Need to calculate the number of blocks evicted during a pre-emption that must be reloaded
• Multiply by the cost to reload each block, BRT
CRPD analysis

• Need to calculate the number of blocks evicted during a pre-emption that must be reloaded
• Multiply by the cost to reload each block, BRT
• Could take a simple approach and assume every block evicted by a pre-empting task must be re-loaded e.g.

\[ \gamma_{t,j} = BRT \Vert ECB_j \]
CRPD analysis

• Adapted a number of approaches for FP to work with EDF

• Defined:
  – the sets of tasks which can/cannot pre-empt each other
  – how often these pre-emption can occur within the interval $t$
CRPD analysis

• Adapted a number of approaches for FP to work with EDF

• Defined:
  – the sets of tasks which can/cannot pre-empt each other
  – how often these pre-emption can occur within the interval $t$

• Then include the CRPD into the $h(t)$ calculation
Integrating CRPD analysis into the $h(t)$ calculation

$$h(t) = \sum_{j=1}^{n} \max \left\{ 0, 1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor \right\} C_j$$
Integrating CRPD analysis into the $h(t)$ calculation

$$h(t) = \sum_{j=1}^{n} \max \left\{ 0, 1 + \left[ \frac{t - D_j}{T_j} \right] \right\} (C_j + \gamma_{t,j})$$

Calculate the CRPD caused by one job of task $\tau_j$ in the interval $t$.
Integrating CRPD analysis into the $h(t)$ calculation

$$h(t) = \sum_{j=1}^{n} \max\left\{ 0, 1 + \left\lfloor \frac{t - D_j}{T_j} \right\rfloor \right\} (C_j + \gamma_{t,j})$$

- Calculate the CRPD caused by one job of task $\tau_j$ in the interval $t$
- Then add it to the execution time of that job of task $\tau_j$
Effect of CRPD on task utilisation and $h(t)$ calculation

• The analysis effectively increases the execution time of a task by the CRPD it causes
• Need to account for this when calculating the utilisation of a task and taskset
• Also need to use this when calculating the upper bound of $t$ used for calculating $h(t)$
Set of pre-empting tasks

• Based on the tasks’ relative deadline
  – Assume that any task $\tau_j$ with a relative deadline $D_j < D_i$ can pre-empt task $\tau_i$
Set of pre-empting tasks

• Based on the tasks’ relative deadline
  – Assume that any task $\tau_j$ with a relative deadline $D_j < D_i$ can pre-empt task $\tau_i$

• The set of tasks that can pre-empt task $\tau_i$ is:
Set of pre-empting tasks

• Based on the tasks’ relative deadline
  – Assume that any task \( \tau_j \) with a relative deadline \( D_j < D_i \) can pre-empt task \( \tau_i \)

• The set of tasks that can pre-empt task \( \tau_i \) is:

\[
hp(i) = \{ \forall \tau_j \mid D_j < D_i \}
\]
Set of pre-empted tasks

- Task $\tau_j$ can pre-empt any tasks whose relative deadline is greater than it’s relative deadline
Set of pre-empted tasks

• Task $\tau_j$ can pre-empt any tasks whose relative deadline is greater than it’s relative deadline

• Can exclude all tasks whose relative deadlines are greater than $t$
  – They do not need to be included when calculating $h(t)$, see paper for proof
Set of pre-empted tasks

• Task $\tau_j$ can pre-empt any tasks whose relative deadline is greater than it’s relative deadline

• Can exclude all tasks whose relative deadlines are greater than $t$
  – They do not need to be included when calculating $h(t)$, see paper for proof

$$\text{aff}(t, j) = \left\{ \forall \tau_i \mid t \geq D_i > D_j \right\}$$
UCB-Union

• Based on approach by Tan and Mooney

\[ \gamma_{t,j}^{ucb-u} = BRT \]
UCB-Union

- Based on approach by Tan and Mooney

\[
\gamma_{t,j}^{\text{ucb-u}} = \text{BRT} \cdot \left( \bigcup_{k \in \text{aff}(t,j)} \text{UCB}_k \right)
\]
UCB-Union

- Based on approach by Tan and Mooney

\[ \gamma_{t,j}^{ucb-u} = BRT \cdot \left( \bigcup_{\forall k \in \text{aff}(t,j)} UCB_k \right) \]

Calculate the union of the UCBs of all tasks that:
- can be evicted by the pre-empting task \( \tau_j \)
- have a job with a release time and absolute deadline within the interval \( t \)
UCB-Union

- Based on approach by Tan and Mooney

\[
\gamma_{t,j}^{ucb-u} = BRT \left| \left( \bigcup_{k \in \text{aff}(t,j)} UCB_k \right) \cap ECB_j \right|
\]

Calculate the union of the UCBs of all tasks that:
- can be evicted by the pre-empting task \( \tau_j \)
- have a job with a release time and absolute deadline within the interval \( t \)

interacted with the ECBs of the pre-empting task
UCB-Union

- Based on approach by Tan and Mooney

\[ \gamma_{t,j}^{ucb-u} = \text{BRT} \cdot \left( \bigcup_{k \in \text{aff}(t,j)} \text{UCB}_k \right) \cap \text{ECB}_j \]

Calculate the union of the UCBs of all tasks that:
- can be evicted by the pre-empting task \( \tau_j \)
- have a job with a release time and absolute deadline within the interval \( t \)

Intersected with the ECBs of the pre-empting task

ECB-Union

• Based on the approach by Altmeyer et al.

\[ \gamma_{t,j}^{ecb-u} = \text{BRT} \]
ECB-Union

• Based on the approach by Altmeyer et al.

\[ \gamma_{t,j}^{ecb-u} = BRT \bigcup_{h \in hp(j) \cup \{j\}} \text{ECB}_h \]

Assume that task \( \tau_j \) has already been pre-empted. Include the union of ECBs belonging to all tasks that can pre-empt it.
ECB-Union

• Based on the approach by Altmeyer et al.

\[ \gamma_{t,j}^{ecb-u} = BRT \bullet \]

\[ \text{UCB}_k \cap \left( \bigcup_{h \in hp(j) \cup \{j\}} \text{ECB}_h \right) \]

Assume that task \( \tau_j \) has already been pre-empted. Include the union of ECBs belonging to all tasks that can pre-empt it.
ECB-Union

• Based on the approach by Altmeyer et al.

\[ \gamma_{t,j}^{ecb-u} = BRT \cdot \max_{\forall k \in \text{aff}(t,j)} \left\{ \text{UCB}_k \cap \left( \bigcup_{h \in h_p(j) \cup \{j\}} \text{ECB}_h \right) \right\} \]

Calculate the maximum number of UCBs that may need to be reloaded by any task that is directly pre-empted by task \( \tau_j \)

Assume that task \( \tau_j \) has already been pre-empted. Include the union of ECBs belonging to all tasks that can pre-empt it
ECB-Union

• Based on the approach by Altmeyer et al.

\[
\gamma_{t,j}^{ecb-u} = BRT \cdot \max \left\{ \bigcup_{k \in \text{aff}(t,j)} \left( \bigcap_{h \in hp(j) \cup \{j\}} \text{UCB}_k \right) \right\}
\]

Calculate the maximum number of UCBs that may need to be reloaded by any task that is directly pre-empted by task \( \tau_j \).

Assume that task \( \tau_j \) has already been pre-empted. Include the union of ECBs belonging to all tasks that can pre-empt it.

Existing approach

• In 2007, Ju et al. presented an approach
  – We refer to it as the JCR approach after their initials

\[ \gamma_{i}^{jcr} = \]
Existing approach

• In 2007, Ju et al. presented an approach
  – We refer to it as the JCR approach after their initials

\[
\gamma_i^{jcr} = \text{BRT} \cdot \sum_{j \in hp(i)}
\]

Sum for every task \(\tau_j\) that can pre-empt task \(\tau_i\)
Existing approach

• In 2007, Ju et al. presented an approach
  – We refer to it as the JCR approach after their initials

\[ \gamma_i^{jcr} = BRT \cdot \sum_{j \in hp(i)} P_j(D_i) \]

Sum for every task \( \tau_j \) that can pre-empt task \( \tau_i \)

the number of times task \( \tau_j \) can pre-empt a single job of task \( \tau_i \)
Existing approach

• In 2007, Ju et al. presented an approach
  – We refer to it as the JCR approach after their initials

\[ \gamma_i^{jcr} = BRT \cdot \sum_{j \in hp(i)} P_j(D_i) |UCB_i \cap ECB_j| \]

- Sum for every task \( \tau_j \) that can pre-empt task \( \tau_i \)
- The number of times task \( \tau_j \) can pre-empt a single job of task \( \tau_i \)
- Multiplied with the number of task \( \tau_i \) UCBs that could be evicted task \( \tau_j \) ECBs
Existing approach

• In 2007, Ju et al. presented an approach
  – We refer to it as the JCR approach after their initials

\[ \gamma_{i}^{jcr} = \text{BRT} \cdot \sum_{j \in \text{hp}(i)} P_{j}(D_{i}) \left| \text{UCB}_{i} \cap \text{ECB}_{j} \right| \]

Sum for every task \( \tau_{j} \) that can pre-empt task \( \tau_{i} \)

the number of times task \( \tau_{j} \) can pre-empt a single job of task \( \tau_{i} \)

multiplied with the number of task \( \tau_{i} \) UCBs that could be evicted task \( \tau_{j} \) ECBs

Existing approach

• Can be pessimistic for nested pre-emption
• Calculates the cost between each pair of tasks
Existing approach

• Can be pessimistic for nested pre-emptions
• Calculates the cost between each pair of tasks
• If pre-empting tasks have ECBs located in the same cache sets, they will be counted multiple times
  – More likely when the cache utilisation is high
Existing approach

- Can be pessimistic for nested pre-emptions
- Calculates the cost between each pair of tasks
- If pre-empting tasks have ECBs located in the same cache sets, they will be counted multiple times
  - More likely when there the cache utilisation is high
- It is incomparable to the approaches we have presented so far
Improved CRPD analysis

- The UCB-Union and ECB-Union approach can be pessimistic
- They assume intermediate tasks are pre-empted the same number of times as the pre-empted task
Improved CRPD analysis

- E.g. the cost of \( \tau_2 \) pre-empting task \( \tau_3 \) is counted three times rather than once.
Improved CRPD analysis

- E.g. the cost of $\tau_2$ pre-empting task $\tau_3$ is counted three times rather than once
Multiset approaches

- ECB-Union Multiset and UCB-Union Multiset
- Factor in the number of times that intermediate tasks pre-empt the pre-empted task to tighten the bound
  - See paper for details
Comparison of approaches

Schedulable
Tasksets
Comparison of approaches

Schedulable Tasksets
Comparison of approaches

Schedulable Tasksets

UCB-U. Multiset  ECB-U. Multiset

UCB-Union  ECB-Union

ECB Only  UCB Only

ECB Only  UCB Only
Comparison of approaches

Schedulable Tasksets

UCB-U. Multiset

ECB-U. Multiset

UCB-Union

ECB-Union

ECB Only

UCB Only

Combined Multiset
Comparison of approaches

Schedulable Tasksets

UCB-U. Multiset

ECB-U. Multiset

UCB-Union

ECB-Union

ECB Only

UCB Only

JCR

Combined Multiset
Case study

• Based on a code from the Mälardalen benchmark suite to create a 15 task taskset

• Setup to model an ARM7
  – 10MHz CPU
  – 2KB direct-mapped instruction cache
  – Line size of 8 Bytes, 4 Byte instructions, 256 cache sets
  – Block reload time of 8μs
## Case study

<table>
<thead>
<tr>
<th></th>
<th>Breakdown utilisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>No pre-emption cost</td>
<td>1</td>
</tr>
<tr>
<td>Combined Multiset</td>
<td>0.659</td>
</tr>
<tr>
<td>ECB-Union Multiset</td>
<td>0.659</td>
</tr>
<tr>
<td>UCB-Union Multiset</td>
<td>0.594</td>
</tr>
<tr>
<td>ECB-Union</td>
<td>0.612</td>
</tr>
<tr>
<td>UCB-Union</td>
<td>0.583</td>
</tr>
<tr>
<td>UCB-Only</td>
<td>0.462</td>
</tr>
<tr>
<td>ECB-Only</td>
<td>0.364</td>
</tr>
<tr>
<td>JCR</td>
<td>0.488</td>
</tr>
</tbody>
</table>
Synthetic tasksets

• 10 tasks per taskset
• 10,000 tasksets for baseline evaluation
• 512 cache sets
• Cache utilisation of 5
• Maximum UCB percentage of 30%
Baseline evaluation
Baseline evaluation
Baseline evaluation

Combined multiset approach outperforms all other approaches including the existing approach.
Baseline evaluation

<table>
<thead>
<tr>
<th>Weighted Schedulability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No pre-emption cost</td>
<td>1</td>
</tr>
<tr>
<td>Combined Multiset</td>
<td>0.528</td>
</tr>
<tr>
<td>ECB-Union Multiset</td>
<td>0.501</td>
</tr>
<tr>
<td>UCB-Union Multiset</td>
<td>0.455</td>
</tr>
<tr>
<td>ECB-Union</td>
<td>0.481</td>
</tr>
<tr>
<td>UCB-Union</td>
<td>0.427</td>
</tr>
<tr>
<td>UCB-Only</td>
<td>0.416</td>
</tr>
<tr>
<td>ECB-Only</td>
<td>0.236</td>
</tr>
<tr>
<td>JCR</td>
<td>0.333</td>
</tr>
</tbody>
</table>
Baseline evaluation

<table>
<thead>
<tr>
<th>Weighted Schedulability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No pre-emption cost</td>
<td>1</td>
</tr>
<tr>
<td>Combined Multiset</td>
<td>0.528</td>
</tr>
<tr>
<td>ECB-Union Multiset</td>
<td>0.501</td>
</tr>
<tr>
<td>UCB-Union Multiset</td>
<td>0.455</td>
</tr>
<tr>
<td>ECB-Union</td>
<td>0.481</td>
</tr>
<tr>
<td>UCB-Union</td>
<td>0.427</td>
</tr>
<tr>
<td>UCB-Only</td>
<td>0.416</td>
</tr>
<tr>
<td>ECB-Only</td>
<td>0.236</td>
</tr>
<tr>
<td>JCR</td>
<td>0.333</td>
</tr>
</tbody>
</table>
Varying Cache Utilisation

![Graph showing varying cache utilisation with different methods compared to the baseline. The y-axis represents the weighted measure, and the x-axis represents the cache utilisation. Various methods are compared, including No Pre-emption Cost, Combined Multiset, ECB-Union Multiset, UCB-Union Multiset, ECB-Union, UCB-Union, UCB-Only, ECB-Only, and JCR.]
Varying Cache Utilisation

![Graph showing varying cache utilisation with different algorithms and measures.](image-url)
Varying Cache Utilisation
Varying Maximum UCB Percentage

Weighted Measure vs. Maximum UCB Percentage

- No Pre-emption Cost
- Combined Multiset
- ECB-Union Multiset
- UCB-Union Multiset
- ECB-Union
- UCB-Union
- ECB-Only
- UCB-Only
- JCR
Varying Maximum UCB Percentage

The graph illustrates the variation in weighted measures as a function of the maximum UCB percentage for different strategies:

- No Pre-emption Cost
- Combined Multiset
- ECB-Union Multiset
- UCB-Union Multiset
- ECB-Union
- UCB-Union
- ECB-Only
- UCB-Only
- JCR

The y-axis represents the weighted measure, and the x-axis represents the maximum UCB percentage.
Varying Maximum UCB Percentage

![Graph showing varying maximum UCB percentage effects](image-url)
Varying Number of Tasks
Conclusion

• Presented new CRPD aware analysis for EDF
• Combined multiset approach dominates the existing approach by Ju et al.
  – Confirmed via evaluation/simulation
• Detailed study shows the strengths and weaknesses of the different approaches
• We plan to investigate which is better, FP or EDF, when taking into account CRPD
Thank you for listening

Any Questions?