Opaque analysis for resource sharing in compositional real-time systems

– 4th CRTS Workshop –

Martijn van den Heuvel*, Moris Behnam, Reinder J. Bril, Johan J. Lukkien and Thomas Nolte

*System Architecture and Networking (SAN)
Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands

29th November 2011
An example for automotive

- Reduce the number of nodes
- Trend: Fewer and more powerful nodes
An example for automotive

- Reduce the number of nodes
- Trend: Fewer and more powerful nodes

**Integration problem:**
- temporal isolation between legacy applications on a single processor;
- applications may share more resources than only the processor.
Hierarchical scheduling frameworks (HSFs)

- Component: server, set of tasks and local (task) scheduler
- Server: a budget allocated each period
Hierarchical scheduling frameworks (HSFs)

- Component: server, set of tasks and local (task) scheduler
- Server: a budget allocated each period

Tasks, located in arbitrary components, may share logical resources
i.e. deal with local and global priority inversion!
Apply SRP at the task level and at the component level:

1. Access shared memory:
   - shared buffers;
   - memory-mapped devices.

2. Operating-system services:
   - in-kernel: short non-preemptive critical sections;
   - device drivers and other services: mutually exclusive between users.

3. Global non-preemptive access to the processor (*pseudo-resources*):
   - reduce cache misses;
   - less pipeline flushes.

**RHTs may be long w.r.t. WCETs!**
Resource holding time (RHT) - assuming $\forall \tau_i : 2P_s \leq T_i$:

- The RHT of task $\tau_i$ to resource $R_l$ is:

  $$X_{sil} = h_{sil} + \sum_{1 \leq j < rc_{sl}} C_j. \quad (1)$$

- The execution time of component $C_s$ to resource $R_l$ is:

  $$X_{sl} = \max\{X_{sil} \mid 1 \leq i \leq n_s\}. \quad (2)$$
Opacity of global resource sharing

A programmer’s perspective:

1. resources are shared between tasks, not between components.

2. tasks claim and protect their resources;

3. Is the resource local or global to a component?
Opacity of global resource sharing

A programmer’s perspective:

1. resources are shared between tasks, not between components.

2. tasks claim and protect their resources;

3. Is the resource local or global to a component? Don’t care!
Opacity of global resource sharing

A programmer’s perspective:

1. resources are shared between tasks, not between components.
2. tasks claim and protect their resources;
3. Is the resource local or global to a component? Don’t care!
   - During the implementation of component code, decouple:
     1. the use of shared resources, and
     2. specification of resources in the interface.
   - postpone binding of SRP primitives until component integration.

P. López Martinez, L. Barros, and J. Drake.
Scheduling configuration of real-time component-based applications.

M. M. H. P. van den Heuvel, R. J. Bril, and J. J. Lukkien.
Transparent synchronization protocols for compositional real-time systems.
IEEE Transactions on Industrial Informatics, PP(99), 2011.

Local view: – resources are available exclusively to a component –
Opacity of global resource sharing

Interface of a component:
- period $P_s$;
- guaranteed budget $Q_s$ within each period;
- set of RHTs $\{X_{s_l}\}$ which defines $X_s = \max\{X_{s_l}\}$.
Opaque resource sharing from an analytical perspective:

1. incremental analysis: separate concerns of local and global scheduling;
2. easier analysis; no timing details on global resource sharing;
3. avoid protocol-specific knowledge to compute $Q_s$. 

Opaque resource sharing from an analytical perspective:

- period $P_s$;
- guaranteed budget $Q_s$ within each period;
- set of RHTs $\{X_{sl}\}$ which defines $X_s = \max_l \{X_{sl}\}$. 
Opacity of global resource sharing

Interface of a component:
- period $P_s$;
- guaranteed budget $Q_s$ within each period;
- set of RHTs $\{X_{sl}\}$ which defines $X_s = \max\{X_{sl}\}$.

Opaque resource sharing from an analytical perspective:
1. incremental analysis: separate concerns of local and global scheduling;
2. easier analysis; no timing details on global resource sharing;
3. avoid protocol-specific knowledge to compute $Q_s$.

Most existing analysis does not support opacity...
Interface of a component:

- period $P_s$;
- guaranteed budget $Q_s$ within each period;
- set of RHTs $\{X_{sl}\}$ which defines $X_s = \max_i \{X_{sl}\}$.

Opaque resource sharing from an analytical perspective:

1. incremental analysis: separate concerns of local and global scheduling;
2. easier analysis; no timing details on global resource sharing;
3. avoid protocol-specific knowledge to compute $Q_s$.

Most existing analysis does not support opacity...

Opacity is a property of the local analysis; not of the interface!
Global resource-sharing problem

*Budget depletion* during a critical section can lead to excessive blocking times:

![Diagram](image_url)
Global resource-sharing problem

*Budget depletion* during a critical section can lead to excessive blocking times:

One of the possible SRP-based solutions for fixed-priority HSFs:

- **HSRP**: React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget.
  - **with payback (OWP)**: the consumed overrun budget is subtracted from the next budget provisioning;
  - **no payback (ONP)**: no penalty for overrun consumption.
Global resource-sharing problem

**Budget depletion** during a critical section can lead to excessive blocking times:

![Diagram showing resource sharing]

One of the possible SRP-based solutions for fixed-priority HSFs:

- **HSRP:** React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget.
  - **1** with payback (OWP): the consumed overrun budget is subtracted from the next budget provisioning;
  - **2** no payback (ONP): no penalty for overrun consumption. → OPAQUE
SRP with overrun: to payback or not?

Observations - pessimism and lack of opacity:

- overrun budget is unnecessary from a local scheduling perspective;
- (c-1): payback increases the processor blackout duration;
- (c-2): payback decreases the supply to (other) tasks.
The effect of payback on the processor blackout

Global admission of component $C_s$:

$$\forall s \exists t \in (0, P_s] : B_s + Q_s + X_s + \sum_{1 \leq r < s} \left( \left\lfloor \frac{t}{P_r} \right\rfloor Q_r \right) + X_r \leq t.$$  

- The latest start time of the processor supply is $P_s - (Q_s + X_s)$.
- Irrespective of an overrun, the blackout remains $2(P_s - Q_s)$. 
The effect of payback on the processor blackout

Enhanced overrun is superfluous:

\[ BD_s = 2(P_s - Q_s) \]

- Reducing the blackout when payback happens by applying a release off-sets for budgets;
- Release off-sets unnecessarily complicate an implementation;
- We accomplish the same without modifying the processor supply.

Unresolved:
- payback decreases the supply to (other) tasks . . .
Given a feasible task set, payback does not affect schedulability

- higher-priority tasks either contribute to overrun or the overrun is not of maximum length.
- blocking of middle-priority tasks starts before overrun.
- lower-priority tasks only experience interference (unchanged).
Opaque, simplified and tighter payback analysis

Under global SRP+FPPS with overrun, a task set is schedulable if

\[ \forall i \exists t \in (0, D_i] : b_i + C_i + \sum_{1 \leq j < i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq \text{sbf}(P_s, Q_s)(t) \]

Observations:

- the same local analysis for both overrun with and without payback;
- the processor supply \text{sbf} is independent of the overrun \( X_s \);
- the local blocking is defined by local SRP;
Opaque, simplified and tighter payback analysis

Under global SRP+FPPS with overrun, a task set is schedulable if

\[ \forall i \exists t \in (0, D_i] : b_i + C_i + \sum_{1 \leq j < i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq sbf(P_s, Q_s)(t) \]

Observations:
- the same local analysis for both overrun with and without payback;
- the processor supply \( sbf \) is independent of the overrun \( X_s \);
- the local blocking is defined by local SRP;

A non-opaque improvement for overrun without payback (IONP):

\[ BD_s = 2(P_s - Q_s) - X_s \]

\( D_s \)
Back to ... the global resource-sharing problem

Budget depletion during a critical section can lead to excessive blocking times:

Two alternative SRP-based solutions for fixed-priority HSFs:

- **HSRP**: React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget;
Budget depletion during a critical section can lead to excessive blocking times:

Two alternative SRP-based solutions for fixed-priority HSFs:

- **HSRP**: React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget;

- **SIRAP**: Prevent budget depletion during resource access; i.e. before granting access, first check the remaining budget.
SIRAP strictly dominates the opaque ONP

Self blocking upon insufficient budget to complete a critical section

Self blocking adds at most $l_{si}$ idle time to the rbf of task $\tau_{si}$ per period $P_s$:

$$l_{si}(t) = \sum_{1 \leq l \leq \left\lfloor \frac{t}{P_s} \right\rfloor} G_{si}^{\text{sort}}(t)[l],$$  \hspace{1cm} (3)

where the sorted multi-set is $G_{si}^{\text{sort}}(t) = \{l_{si}^{\text{low}}\} \cup$

$$\bigcup_{1 \leq j \leq i} \bigcup_{R_l \in \mathcal{R}_s} \bigcup_{1 \leq k \leq \left\lfloor \frac{t}{T_{sj}} \right\rfloor} \bigcup_{1 \leq a \leq m_{sjl}} \{X_{sjl}\}. \hspace{1cm} (4)$$

- Contrary to overrun: SIRAP computes an upper bound to the number of resource accesses;
SIRAP strictly dominates the opaque ONP

Self blocking upon insufficient budget to complete a critical section

Self blocking adds at most $l_{si}$ idle time to the $rbf$ of task $\tau_{si}$ per period $P_s$:

$$l_{si}(t) = \sum_{1 \leq l \leq \left\lfloor \frac{t}{P_s} \right\rfloor} G_{si}^{sort}(t)[l],$$  \hspace{1cm} (3)$$

where the sorted multi-set is $G_{si}^{sort}(t) = \{l_{si}^{low}\} \cup$

$$\bigcup_{(1 \leq j \leq i)} \bigcup_{(R_l \in R_s)} \bigcup_{(1 \leq k \leq \left\lceil \frac{t}{T_{sj}} \right\rceil)} \bigcup_{(1 \leq a \leq m_{sjl})} \{X_{sjl}\}.$$  \hspace{1cm} (4)$$

- Contrary to overrun: SIRAP computes an upper bound to the number of resource accesses;
- Each individual access is at most of length $X_s$. 

Martijn van den Heuvel (TU/e, SAN)  Opaque analysis for resource sharing  29th November 2011  15 / 20
Abstraction overhead due to opaque analysis

- generate tasks with UUnifast and periods in \([140, 1000]\);
- critical section lengths between \(0.1 \times C_i\) and \(0.25 \times C_i\);
- the number of tasks is \(n_s = 8\);
- 1000 samples: count the number of feasible components

Component period is \(P_s = 40\).

Component utilization is 0.4.
Abstraction overhead due to opaque analysis

Component period is $P_s = 40$.

Component utilization is 0.4.

Constraints:
- Overrun with and without payback: $Q_s + X_s \leq P_s$;
- SIRAP: $X_s \leq Q_s$. 
Abstraction overhead due to opaque analysis

Component period is $P_s = 40$.

Component utilization is 0.4.

Constraints:

- Overrun with and without payback: $Q_s + X_s \leq P_s$;
- SIRAP: $X_s \leq Q_s$.

The ratio $\frac{X_s}{P_s}$ is the main weakness of overrun.
Abstraction overhead due to opaque analysis

• With global fixed-priority scheduling, opacity has its price.

• Our contributions:

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Opacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSRP - ONP (D&amp;B 2006)</td>
<td>no</td>
</tr>
<tr>
<td>HSRP - ONP (Behnam et al. 2010)</td>
<td>yes</td>
</tr>
<tr>
<td>HSRP - OWP</td>
<td>yes</td>
</tr>
<tr>
<td>Enhanced overrun</td>
<td>no</td>
</tr>
<tr>
<td>Improved overrun without payback (IONP)</td>
<td>no</td>
</tr>
<tr>
<td>SIRAP</td>
<td>with opaque ONP</td>
</tr>
</tbody>
</table>
Abstraction overhead due to opaque analysis

- With global fixed-priority scheduling, opacity has its price.

- **Our contributions:**

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Opacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSRP - ONP (D&amp;B 2006)</td>
<td>no</td>
</tr>
<tr>
<td>HSRP - ONP (Behnam et al. 2010)</td>
<td>yes</td>
</tr>
<tr>
<td>HSRP - OWP</td>
<td>yes</td>
</tr>
<tr>
<td>Enhanced overrun</td>
<td>no</td>
</tr>
<tr>
<td>Improved overrun without payback (IONP)</td>
<td>no</td>
</tr>
<tr>
<td>SIRAP</td>
<td>with opaque ONP</td>
</tr>
</tbody>
</table>

- **A solution** - incremental analysis:
  - developers apply an opaque analysis to their component;
  - integrators apply refined analysis to each individual component.
With global fixed-priority scheduling, opacity has its price.

Open question: are there any alternatives?

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Opacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROE</td>
<td>yes?</td>
</tr>
<tr>
<td>HSRP - ONP (D&amp;B 2006)</td>
<td>no</td>
</tr>
<tr>
<td>HSRP - ONP (Behnam et al. 2010)</td>
<td>yes</td>
</tr>
<tr>
<td>HSRP - OWP (D&amp;B 2006, Behnam et al. 2010)</td>
<td>no</td>
</tr>
<tr>
<td>Enhanced overrun</td>
<td>no</td>
</tr>
<tr>
<td>Improved overrun without payback (IONP)</td>
<td>no</td>
</tr>
<tr>
<td>SIRAP</td>
<td>no</td>
</tr>
</tbody>
</table>

Future work: how does BROE perform?
- uses global EDF.
Conclusions and recommendations

An opaque analysis at the task level defers the choice for a global resource-arbitration policy until component integration time.

We contributed:

- an opaque analysis for overrun with payback (OWP);
- a tighter and simplified analysis for OWP;
- an opaque analysis for SIRAP using overrun;
- a comparison of analysis methods.
Conclusions and recommendations

An opaque analysis at the task level defers the choice for a global resource-arbitration policy until component integration time.

We contributed:

- an opaque analysis for overrun with payback (OWP);
- a tighter and simplified analysis for OWP;
- an opaque analysis for SIRAP using overrun;
- a comparison of analysis methods.

Guidelines for choosing a protocol:

- In most cases OWP is more efficient than ONP;
- Enhanced overrun is superfluous, i.e. outperformed by OWP;
- SIRAP strictly outperforms ONP (and often OWP).