Constant Bandwidth Servers with Constrained Deadlines

Daniel Casini, Luca Abeni, Alessandro Biondi, Tommaso Cucinotta, and Giorgio Buttazzo

Scuola Superiore Sant’Anna – ReTiS Laboratory

Pisa, Italy
This talk in a nutshell

1. Challenges in designing a reservation servers with constrained deadlines
2. Three new different algorithms
3. Simulation study to assess their performance
Why using constrained-deadlines?

Recent work showed that Semi-Partitioned scheduling can achieve high schedulability performance with low complexity:

- “Global Scheduling Not Required” by Brandenburg and Gul for static workloads (RTSS 2016)
- “Semi-Partitioned Scheduling of Dynamic Real-Time Workload” by Casini et al. for dynamic workloads (ECRTS 2017)
Why using constrained-deadlines?

Recent work showed that Semi-Partitioned scheduling can achieve high schedulability performance with low complexity:

- “Global Scheduling Not Required” by Brandenburg and Gul for static workloads (RTSS 2016)
- “Semi-Partitioned scheduling of Dynamic Real Time Workload” by Casini et al. for dynamic workloads (ECRTS 2017)

Both requires constrained-deadline (C=D) reservations!
Why using constrained-deadlines?

- Supporting constrained-deadlines is an open problem also for the `SCHED_DEADLINE` scheduling class of Linux (based on reservations with the H-CBS algorithm).

- Currently discussed also in the Linux kernel mailing list.
Hard Constant Bandwidth Server

- H-CBS is a reservation algorithm allowing to guarantee:

  - A bandwidth $\alpha = \frac{Q}{P}$
  
  - A bounded maximum service-delay $\Delta = 2(P - Q)$

Worst-case scenario for the service delay

\[ \Delta = 2(P - Q) \]
Importance of a bounded delay

A bounded-delay allows deriving a supply-bound function that can be used for testing the schedulability of the workload running inside the server:

Case of implicit-deadlines

\[ \text{sbf}(t) \]
H-CBS key rule

H-CBS has a specific rule to generate a new budget and scheduling deadline when the server wakes up from the idle state:

\[
(q, d) = \begin{cases} 
(q, d) & \text{if } t < \frac{d - q}{\alpha} \\
(Q, t + P) & \text{otherwise}
\end{cases}
\]

This rule has been derived by EDF schedulability theory for implicit-deadline tasks (utilization-based), which indeed cannot be re-used to ensure schedulability with constrained deadlines!
Possible simple solutions

- Mimic the polling server

- Higher worst-case delay!

\[ \Delta = P + (D - Q) \]

- Configure H-CBS to use D in place of P

- High pessimism!

- \( S_{C=D} \) 100%

CPU
Design Issues

How to modify the server rules for providing a given bandwidth and a better maximum-service delay, without sacrificing schedulability?

How to achieve a maximum service delay equal to $\Delta = D + P - 2Q$?
Our Solutions

- **H-CBS\textsuperscript{D}-W** (H-CBS Deadline – Worst Case)
  - Our solution for hard real-time systems

- **H-CBS\textsuperscript{D}** (H-CBS Deadline)
  - Our solution to improve average-case performance for soft real-time systems

- **H-CBS\textsuperscript{D}-R** (H-CBS Deadline - Reclaiming)
  - Extends H-CBS\textsuperscript{D} with reclaiiming
**H-CBS\textsuperscript{D}-W**: Idea

- **H-CBS\textsuperscript{D}-W** leverages the results proposed by Biondi et al. for real-time self-suspending tasks


- According to their approach, whenever a server should execute according to EDF scheduling, it consumes its budget independently whether it is self-suspended or not
A similar approach can be adopted when a reservation goes idle:

- Server goes idle

\( S_1 \) consumes its budget even if it is idle.
**H-CBS^D-W : Evaluation**

**PROS:**

- It guarantees a bandwidth \( \alpha \) and a bounded delay \( \Delta = D + P - 2Q \)

- H-CBS^D-W is independent from the adopted schedulability test!

**CONS:**

- It requires the implementation of an additional server queue to keep track of suspended servers

Room for improvement in terms of average-case performance and soft real-time metrics
CAN WE DO BETTER (IN TERMS OF AVERAGE-CASE PERFORMANCE AND SOFT REAL-TIME METRICS) BY LEVERAGING A SPECIFIC SCHEDULABILITY ANALYSIS?

THE $H - CBS^D$ ALGORITHM
**H-CBS$^D$ Idea**

- Assign current budget and deadline by means of an online schedulability test based on approximated demand bound functions, but...

Leveraging also the knowledge of online parameters!

- The current budget $q_i$
- The scheduling deadline $d_i$
From the knowledge of such online variables, we derived a schedulability test based on the following abstraction of the workload:

**Run-time demand function**

\[ rdf_i(t, t^*, q_i, d_i) \]

- Time in which the schedulability test is performed (at run-time)
**H-CBS^D** key rule

- **H-CBS^D** assigns a new **budget and scheduling deadline** when the server **wakes up** from the idle state based on the following idea:

\[
\begin{align*}
\text{re-use } (q, d) \\
q &= Q \text{ with } d = t + D
\end{align*}
\]

- **Similar to H-CBS** but using a schedulability test for valid **constrained-deadline servers**

- If the rdf-based test holds, use \((q, d)\).
- Otherwise, use \(q = Q \) with \(d = t + D\).
**H-CBS\textsuperscript{D}**: Main Properties

- **Bounded-delay**: the algorithm guarantees a bounded worst-case service delay
  \[ \Delta = D + P - 2Q \]

- **Reserves** a portion of the processor capacity for idle reservations

- A **full budget replenishment** is guaranteed to each reservation after at least \( P_i \) units of time from the last replenishment
A different approach: \(H-CBS^D-R\)

**Idea:** Let’s allow a reservation to take the maximum possible budget which does not break schedulability!

- A reservation adopting \(H-CBS^D-R\) implicitly implements a budget reclaiming.
- Average-case performance and probabilistic metrics could benefit of this approach. (see simulations results...following slides!)
The $H-CBS^D-R$ algorithm

How to do this?

- We leveraged the $rdf$-based schedulability analysis developed for $H - CBS^D$ to derive a sensitivity analysis:

  **Find:** \( \max q \) such that schedulability is not violated
SIMULATION RESULTS

http://retis.santannapisa.it/~luca/RTNS17
Reservation generation

- First, we generated reservation bandwidths with the Emberson et al. Task-set generator, with:
  - Periods distributed in [5000; 500000] us
  - Budget obtained as $Q_i = \alpha_i P_i$
  - Relative deadline generated with uniform distribution in $[Q_i + \beta (P_i - Q_i), P_i]$

- The workload running into each reservation consist of a single sporadic task
  - Reservation-set with results to be unschedulable according to the approximated schedulability test have been discarded
  - 100 different reservation-set have been tested
Workload generation

- Each job running into a reservation (i.e. its computation and inter-arrival time) is controlled by:
  - $C_r = \frac{c_i}{Q_i}$
  - $a = \frac{u_i}{\alpha_i}$, with $\alpha_i = \frac{Q_i}{P_i}$, and $\bar{u}_i = \frac{c_i}{P_i}$

- Variance of execution (sc) and inter-arrival times (sp)
  - The execution time of each job is uniformly distributed in $[c_i - \frac{sc(sz)}{2}, c_i + \frac{sc(sz)}{2}]$
  - The inter-arrival time of each job is uniformly distributed in $[p_i - \frac{sp(sz)}{2}, p_i + \frac{sp(sz)}{2}]$
Workload generation

- Each job running into a reservation (i.e. its computation and inter-arrival time) is controlled by:
  - \( C_r = \frac{c_i}{Q_i} \)
  - \( a = \frac{u_i}{\alpha_i} \), with \( \alpha_i = \frac{Q_i}{P_i} \), and \( u_i = \frac{c_i}{p_i} \)

- Variance of execution (sc) and inter-arrival times (sp)
  - The execution time of each job is uniformly distributed in
    \( [\bar{c}_i - \frac{sc(sz)}{2}, \bar{c}_i + \frac{sc(sz)}{2}] \)
  - The inter-arrival time of each job is uniformly distributed in
    \( [\bar{p}_i - \frac{sp(sz)}{2}, \bar{p}_i + \frac{sp(sz)}{2}] \)
Workload generation

- Each job running into a reservation (i.e. its computation and inter-arrival time) is controlled by:

  \[ C_r = \frac{\bar{c}_i}{Q_i} \]

  - Ratio between the average task utilization and the server bandwidth

  \[ a = \frac{\bar{u}_i}{\alpha_i} \] with \( \alpha_i = \frac{Q_i}{P_i} \), and \( \bar{u}_i = \frac{\bar{c}_i}{P_i} \)

- Variance of execution (sc) and inter-arrival times (sp)

  - The execution time of each job is uniformly distributed in

    \[ [\bar{c}_i - \frac{sc(sz)}{2}, \bar{c}_i + \frac{sc(sz)}{2}] \]

  - The inter-arrival time of each job is uniformly distributed in

    \[ [\bar{p}_i - \frac{sp(sz)}{2}, \bar{p}_i + \frac{sp(sz)}{2}] \]
Workload generation

- Each job running into a reservation (i.e. its computation and inter-arrival time) is controlled by:

  - $C_r = \frac{c_i}{Q_i}$
  
  - $a = \frac{u_i}{\alpha_i}$, with $\alpha_i = \frac{Q_i}{P_i}$, and $u_i = \frac{c_i}{p_i}$

- Variance of execution (sc) and inter-arrival times (sp)

  - The execution time of each job is uniformly distributed in $[\bar{c}_i - \frac{sc(sz)}{2}, \bar{c}_i + \frac{sc(sz)}{2}]$
  
  - The inter-arrival time of each job is uniformly distributed in $[\bar{p}_i - \frac{sp(sz)}{2}, \bar{p}_i + \frac{sp(sz)}{2}]$

Variances are function of the sz generation parameter
Simulation Results

The higher the better

Relative server workload

sz=0.8 cr=0.6 U=0.7

deadline hit ratio

H-CBS\textsuperscript{D-W}
H-CBS\textsuperscript{D}
H-CBS\textsuperscript{D-R}

a
Simulation Results

The higher the better

$H^{-CBS^D-R}$ has up to 60% of improvement over $H^{-CBS^D-W}$
Simulation Results

The higher the better

Relative server workload

$H - CBS^D - R$ has up to 25% of improvement over $H - CBS^D$

$H - CBS^D - R$ has up to 60% of improvement over $H - CBS^D - W$
Simulation Results

The higher the better

deadline hit ratio (a=0.60, U=0.7)

H-CBS\textsuperscript{D-W}

H-CBS\textsuperscript{D}

H-CBS\textsuperscript{D-R}
Simulation Results

deadline hit ratio \((a=0.60, \ U=0.7)\)

- H-CBS\(^{D-W}\)
- H-CBS\(^{D}\)
- H-CBS\(^{D-R}\)

Task/Server computation time ratio
Simulation Results

deadline hit ratio \( (a=0.60, U=0.7) \)

H-CBS\(^D\)-W
H-CBS\(^D\)
H-CBS\(^D\)-R

Variance of computation and inter-arrival times
Simulation Results

deadline hit ratio (a=0.60, U=0.7)

Similar trend
**Comparison**

<table>
<thead>
<tr>
<th>H−CBS^{D}−W</th>
<th>H−CBS^{D} &amp; H−CBS^{D}−R</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅ Guarantees a given bandwidth</td>
<td>✅ Guarantees a given bandwidth</td>
</tr>
<tr>
<td>✅ Bounded delay ( \Delta = D + T - 2Q )</td>
<td>✅ Bounded delay ( \Delta = D + T - 2Q )</td>
</tr>
<tr>
<td>✅ Wake-up budget can be computed in constant time</td>
<td>✅ Improves soft-real time metrics</td>
</tr>
<tr>
<td>✅ Independent from the schedulability test</td>
<td>✗ Wake-up budget can be computed in linear time</td>
</tr>
<tr>
<td>✅ Compatible also with global scheduling</td>
<td>✗ Not compatible with global scheduling</td>
</tr>
<tr>
<td>✗ Requires an additional queue for suspended servers</td>
<td>✗ Penalties in terms of schedulability (sufficient test)</td>
</tr>
<tr>
<td>✗ Consumes budget also without actually executing</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- We proposed three different reservations servers supporting constrained-deadlines

- All of them allow to provide a given bandwidth and a bounded delay

- $H^{CBS^D}_W$ is independent from the schedulability test

- Suitable for hard real-time systems

- $H^{CBS^D}$ and $H^{CBS^D}_R$ leverages a specific schedulability test to improve soft real-time metrics

- The performance of the proposed algorithms has been experimented by means of simulations
Future Work

- Include a reclaiming mechanism in $H-CBS^D-W$ to improve average-case performance
- Extended our solutions to cope with shared resources
- Implementation in a real-time operating systems (e.g., Linux under `SCHED_DEADLINE`)
Thank you!

Daniel Casini
daniel.casini@sssup.it