SCHED_DEADLINE: It’s Alive!

Juri Lelli
ARM Ltd.

ELC North America 17, Portland (OR)
02/21/2017

©ARM 2017
Agenda

- Deadline scheduling (SCHED_DEADLINE)
- Why is development now happening (out of the blue?)
- Bandwidth reclaiming
- Frequency/CPU scaling of reservation parameters
- Coupling with frequency selection
- Group scheduling
- Future
CHAPTER I
What and Why
Agenda

- Deadline scheduling (SCHED_DEADLINE)
- Why is development now happening (out of the blue?)
- Bandwidth reclaiming
- Frequency/CPU scaling of reservation parameters
- Coupling with frequency selection
- Group scheduling
- Future
Deadline scheduling (previously on ...)

- mainline since v3.14
  30 March 2014 (~3y ago)
- it’s not only about deadlines
  - RT scheduling policy
  - explicit per-task latency constraints
  - avoids starvation
  - enriches scheduler’s knowledge about QoS requirements
- EDF + CBS
- resource reservation mechanism
- temporal isolation
- ELC16 presentation
  https://goo.gl/OVspuI
Deadline scheduling (previously on ...)

- mainline since v3.14
  30 March 2014 (~3y ago)
- it’s not only about deadlines
  - RT scheduling policy
  - explicit per-task latency constraints
  - avoids starvation
  - enriches scheduler’s knowledge about QoS requirements
- EDF + CBS
- resource reservation mechanism
- temporal isolation
- ELC16 presentation
  https://goo.gl/OVspuI
Agenda

- Deadline scheduling (SCHED_DEADLINE)
- Why is development now happening (out of the blue?)
- Bandwidth reclaiming
- Frequency/CPU scaling of reservation parameters
- Coupling with frequency selection
- Group scheduling
- Future
Why is development now happening

- **Energy Aware Scheduling (EAS)**
  - extends the Linux kernel scheduler and power management to make it fully power/performance aware ([https://goo.gl/vQbUOu](https://goo.gl/vQbUOu))
  - scheduler modifications pertain to SCHED_NORMAL (so far)

- **Android Common Kernel**
  - EAS has been merged last year ([https://goo.gl/FXCdAX](https://goo.gl/FXCdAX))
  - performance usually means meeting latency requirements
  - considerable usage (and modifications) of SCHED_FIFO
  - SCHED_DEADLINE seems to be a better fit
  - and mainline adoption of required changes *should be* less controversial

- Joint collaboration between ARM and Scuola Superiore Sant’Anna of Pisa
CHAPTER 2
Let’s reclaim!
Agenda

- Deadline scheduling (SCHED_DEADLINE)
- Why is development now happening (out of the blue?)
- Bandwidth reclaiming
- Frequency/CPU scaling of reservation parameters
- Coupling with frequency selection
- Group scheduling
- Future
Bandwidth Reclaiming

- **PROBLEM**
  - tasks’ bandwidth is fixed (can only be changed with `sched_setattr()`)
  - what if tasks occasionally need more bandwidth? e.g., occasional workload fluctuations (network traffic, rendering of particularly heavy frame, etc)

- **SOLUTION (proposed*)**
  - bandwidth reclaiming: allow tasks to consume more than allocated
  - up to a certain maximum fraction of CPU time
  - if this doesn’t break others’ guarantees

* [https://lkml.org/lkml/2016/12/30/107](https://lkml.org/lkml/2016/12/30/107)
Bandwidth Reclaiming (cont.)

- Greedy Reclamation of Unused Bandwidth (GRUB)\(^1\)
- 3 components\(^2\)
  - tracking of active utilization
  - modification of the accounting rule
  - multiprocessor support (original algorithm was designed for UP)

\(^1\) Greedy reclamation of unused bandwidth in constant-bandwidth servers - Giuseppe Lipari, Sanjoy K. Baruah (https://goo.gl/xl4CUk)
\(^2\) Greedy CPU reclaiming for SCHED DEADLINE - Luca Abeni, Juri Lelli, Claudio Scordino, Luigi Palopoli (https://goo.gl/e8EC8q)
Bandwidth Reclaiming (cont.)

- Tracking of active utilization

- Uact is increased by Qi/Ti when task wakes up
- 0 lag time comes from CBS wakeup check: \( \frac{q_i}{d_i - t} < \frac{Q_i}{T_i} \)
- Uact is decreased by the same amount at 0 lag time, a timer is set to fire at this instant of time
- One Uact per CPU (rq->dl.running_bw)
Bandwidth Reclaiming (cont.)

- Tracking of active utilization

- Uact is increased by \( \frac{Q_i}{T_i} \) when task wakes up
- 0 lag time comes from CBS wakeup check: \( \frac{q_i}{d_i - t} < \frac{Q_i}{T_i} \)
- Uact is decreased by the same amount at 0 lag time
  a timer is set to fire at this instant of time
- One Uact per CPU \((rq->dl.running_bw)\)
Bandwidth Reclaiming (cont.)

- Modification of the accounting rule

- \( \text{runtime} = \text{delta}\_\text{exec} \) becomes \( \text{runtime} = U_{\text{act}} \times \text{delta}\_\text{exec} \)
- but this can eat up 100% of CPU time! (starving non-DL tasks)
- e.g., a 5sec every 10sec task that can reclaim...
- so, in reality accounting will probably become
  \( \text{runtime} = \frac{U_{\text{act}}}{U_{\text{max}}} \times \text{delta}\_\text{exec} \)
Bandwidth Reclaiming (cont.)

- Modification of the accounting rule

  - $\text{runtime} \:= \text{delta}_{\text{exec}} \quad \Rightarrow \quad \text{runtime} \:= \frac{U_{\text{act}}}{U_{\text{max}}} \cdot \text{delta}_{\text{exec}}$

  - but this can eat up 100% of CPU time! (starving non-DL tasks)

  - e.g., a 5sec every 10sec task that can reclaim...

  - so, in reality accounting will probably become $\text{runtime} \:= \frac{U_{\text{act}}}{U_{\text{max}}} \cdot \text{delta}_{\text{exec}}$
Bandwidth Reclaiming (cont.)

- e.g., a 5sec every 10sec task that can’t reclaim...

- VS, a 5sec every 10sec task that can reclaim (without Umax cap)

\[
U_{\text{act}} = 0.5 \rightarrow \text{runtime} = \delta \cdot 0.5 \rightarrow \text{deplete in} \ (1/0.5) \cdot \text{runtime} = 10\text{sec}
\]

\[
U_{\text{max}} = 0.9 \rightarrow \text{runtime} = \delta \cdot (0.5/0.9) \rightarrow \text{deplete in} \ (0.9/0.5) \cdot \text{runtime} = 9\text{sec}
\]

leaving 1sec for otherwise sad guys :-)
Bandwidth Reclaiming (cont.)

- Multiprocessor support
- ISSUE (one of a few)

\[ U_{act_k} + \frac{Q_i}{T_i} = 0_{lag} \]

- task \( i \) wakes up and is accounted for
- it then blocks and timer is set to fire at 0 lag time
Bandwidth Reclaiming (cont.)

- Multiprocessor support
- ISSUE (one of a few)

- task \( i \) wakes up again, before 0 lag
- but it is migrated on a different CPU
- 0 lag timer cancelled, but no changes to both CPUs’ Uact
Bandwidth Reclaiming (cont.)

- Multiprocessor support
- ISSUE (one of a few)

- task $i$ blocks again (on CPU$j$)
- no change on CPU$k$’s $U_{act}$ and CPU$j$’s $U_{act}$ becomes negative!

$$
U_{act_k}(t-1) = U_{act_k}(t)
$$

$$
U_{act_j} = \frac{Q_i}{T_i} < 0
$$
Bandwidth Reclaiming (cont.)

- Multiprocessor support
- SOLUTION – migrate task’s utilization together with him

- 0 lag timer cancelled, and...
- utilization is instantaneously migrated as well
- so that when task i blocks again everything is fine
Bandwidth Reclaiming (results)

- Task1 (6ms, 20ms) constant execution time of 5ms
- Task2 (45ms, 260ms) experiences occasional variances (35ms-52ms)
Bandwidth Reclaiming (results)

- Task1 (6ms, 20ms) constant execution time of 5ms
- Task2 (45ms, 260ms) experiences occasional variances (35ms-52ms)

- Cumulative Distribution Function (CDF) probability that Response time will be less or equal to x ms
Bandwidth Reclaiming (results)

- Task1 (6ms, 20ms) constant execution time of 5ms
- Task2 (45ms, 260ms) experiences occasional variances (35ms-52ms)
- Plain CBS T2’s response time bigger than reservation period (~25%)
Bandwidth Reclaiming (results)

- Task 1 (6ms, 20ms) constant execution time of 5ms
- Task 2 (45ms, 260ms) experiences occasional variances (35ms-52ms)

- GRUB
  T2 always completes before reservation period (using bandwidth left by T1)
CHAPTER 3
Rock around the Clock (... and CPU)
Agenda

- Deadline scheduling (SCHED_DEADLINE)
- Why is development now happening (out of the blue?)
- Bandwidth reclaiming
- Frequency/CPU scaling of reservation parameters
- Coupling with frequency selection
- Group scheduling
- Future
Frequency/CPU scaling

- Reservation runtime needs scaling according to frequency and CPU max capacity
- for frequency, use the ratio between max and current capacity to enlarge the runtime granted to a task at admission control

\[ \text{scaled \_runtime} = \text{original \_runtime} \cdot \frac{\text{max \_capacity}}{\text{curr \_capacity}} \]

- similarly for CPU, but using the ratio between biggest and current CPU capacity
Frequency scaling (example)

- HiKey board has 5 Operating Performance Points (OPPs)

<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Capacity</th>
<th>% w.r.t. max</th>
</tr>
</thead>
<tbody>
<tr>
<td>208</td>
<td>178</td>
<td>17</td>
</tr>
<tr>
<td>432</td>
<td>369</td>
<td>36</td>
</tr>
<tr>
<td>729</td>
<td>622</td>
<td>61</td>
</tr>
<tr>
<td>960</td>
<td>819</td>
<td>80</td>
</tr>
<tr>
<td>1200</td>
<td>1024</td>
<td>100</td>
</tr>
</tbody>
</table>

- Running a task inside a 12ms/100ms reservation at min frequency means extending its runtime up to

\[
scaled \_\text{runtime} = 12ms \cdot \frac{1024}{178} \approx 69ms
\]
Frequency scaling (example cont.)

- 10ms/100ms task inside a 12ms/100ms reservation (at max freq)

- 10ms/100ms task inside a 12ms/100ms reservation (at min freq)

- 20ms/100ms (bad guy!) task inside a 12ms/100ms reservation (at min freq)
Agenda

- Deadline scheduling (SCHED_DEADLINE)
- Why is development now happening (out of the blue?)
- Bandwidth reclaiming
- Frequency/CPU scaling of reservation parameters
- **Coupling with frequency selection**
- Group scheduling
- Future
Driving frequency selection

- scaling clock frequency, while meeting tasks’ requirements (deadlines)
- scheduler driven CPU clock frequency selection
  - schedutil cpufreq governor
    SCHED_NORMAL – uses util_avg (PELT)
    SCHED_FIFO/RR and SCHED_DEADLINE – go to max!

- once bandwidth reclaiming is in*
  - use rq->dl.running_bw as SCHED_DEADLINE per-CPU utilization contribution (sum)
  - move CPU frequency selection triggering points (where running_bw changes)
  - allow sugov kworker thread(s) to always preempt SCHED_DEADLINE tasks (and lower priority) – for !fast_switch_enabled drivers

* Claudio Scordino (Evidence Srl) is helping with this.
Driving frequency selection

- scaling clock frequency, while meeting tasks’ requirements (deadlines)
- scheduler driven CPU clock frequency selection
  - schedutil cpufreq governor
    - SCHED_NORMAL – uses util_avg (PELT)
    - SCHED_FIFO/RR and SCHED_DEADLINE – go to max!

- once bandwidth reclaiming is in*
  - use rq->dl.running_bw as SCHED_DEADLINE per-CPU utilization contribution (sum)
  - move CPU frequency selection triggering points (where running_bw changes)
  - allow sugov kworker thread(s) to always preempt SCHED_DEADLINE tasks (and lower priority) – for !fast_switch_enabled drivers

* Claudio Scordino (Evidence Srl) is helping with this.
Driving frequency selection (example)

- 50ms/100ms inside 52ms/100ms + 10ms/100ms inside 12ms/100ms
- rt-app\(^1\) based measure of “performance”

\[
\text{perf\_index} = \frac{\text{measured\_slack}}{\text{config\_slack}}
\]

- \(\text{perf\_index}\) close to 1.0 means almost optimal performance
- negative \(\text{perf\_index}\) means deadline misses

\(^1\) - https://github.com/scheduler-tools/rt-app
Driving frequency selection (example)

- 50ms/100ms inside 52ms/100ms + 10ms/100ms inside 12ms/100ms

- deadlines are not missed while frequency is not at max (960MHz mostly)

complete set of results available at https://gist.github.com/jlelli/22196e46e4ff1fcd02a9944261d90d2
CHAPTER 4

Groupies
Agenda

- Deadline scheduling (SCHED_DEADLINE)
- Why is development now happening (out of the blue?)
- Bandwidth reclaiming
- Frequency/CPU scaling of reservation parameters
- Coupling with frequency selection
- Group scheduling
- Future
Group scheduling

- Currently, one to one association between tasks and reservations
- Sometime it might be better/easier to group a set of tasks into the same reservation
  - virtual machine threads
  - rendering pipeline
  - legacy application (that for example needs forking)
  - high priority driver kthread(s)
- Hierarchical/Group scheduling\(^1,2,3\)
  - cgroups support
  - temporal isolation between groups (and single entities)

3. The IRMOS real-time scheduler - T. Cucinotta, F. Checconi (https://lwn.net/Articles/398470/)
Group scheduling

- Hierarchical means
  - first level is EDF
  - second level is RT (FIFO/RR)

- Should eventually supplant RT-throttling
Group scheduling

- Hierarchical means
  - first level is EDF
  - second level is RT (FIFO/RR)

- Should eventually supplant RT-throttling

![Diagram showing group scheduling with EDF at the top, FIFOs below it, and tasks T1 to T4 distributed across the levels.]

QUIZ
Group scheduling

- On multiprocessors

- One DEADLINE group entity per CPU
- Coexists with single DEADLINE entities
Group scheduling

- On multiprocessors
  - One DEADLINE group entity per CPU
  - Coexists with single DEADLINE entities
  - Sub RT entities get migrated according to G-FP (push/pull)
CHAPTER 5
It IS bright!
Agenda

- Deadline scheduling (SCHED_DEADLINE)
- Why is development now happening (out of the blue?)
- Bandwidth reclaiming
- Frequency/CPU scaling of reservation parameters
- Coupling with frequency selection
- Group scheduling
- Future
Future

- NEAR
  - experimenting with Android
  - reclaiming by demotion towards lower priority class
  - capacity awareness (for heterogeneous systems)
  - energy awareness (Energy Aware Scheduling for DEADLINE)

- NEAR(...ISH)
  - support single CPU affinity
  - enhanced priority inheritance (M-BWI most probably)
  - dynamic feedback mechanism (adapt reservation parameters to task’ needs)
Get involved!

Shoot me an email <juri.lelli@arm.com>
Ask questions on LKML, linux-rt-users or eas-dev
Come join us @ OSPM-summit (https://goo.gl/ngTcgB)
... maybe remotely :-)

And don’t forget to collect your prizes!!!

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

Copyright © 2017 ARM Limited

©ARM 2017

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

Copyright © 2017 ARM Limited

©ARM 2017