SCHED_DEADLINE: What's next?

Claudio Scordino, Evidence Srl
Juri Lelli, Red Hat

Outline

- Already mainline:
  - Bandwidth reclaiming (GRUB)
- On-going development:
  - Schedutil integration (GRUB-PA)
  - Hierarchical/group scheduling
  - Semi-partitioned scheduling
- Under discussion:
  - Reclaiming by demotion
  - Throttled signaling
  - (Single CPU) affinity
  - Unprivileged usage
  - Proxy execution/M-BWI

Bandwidth reclaiming (GRUB)

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.)

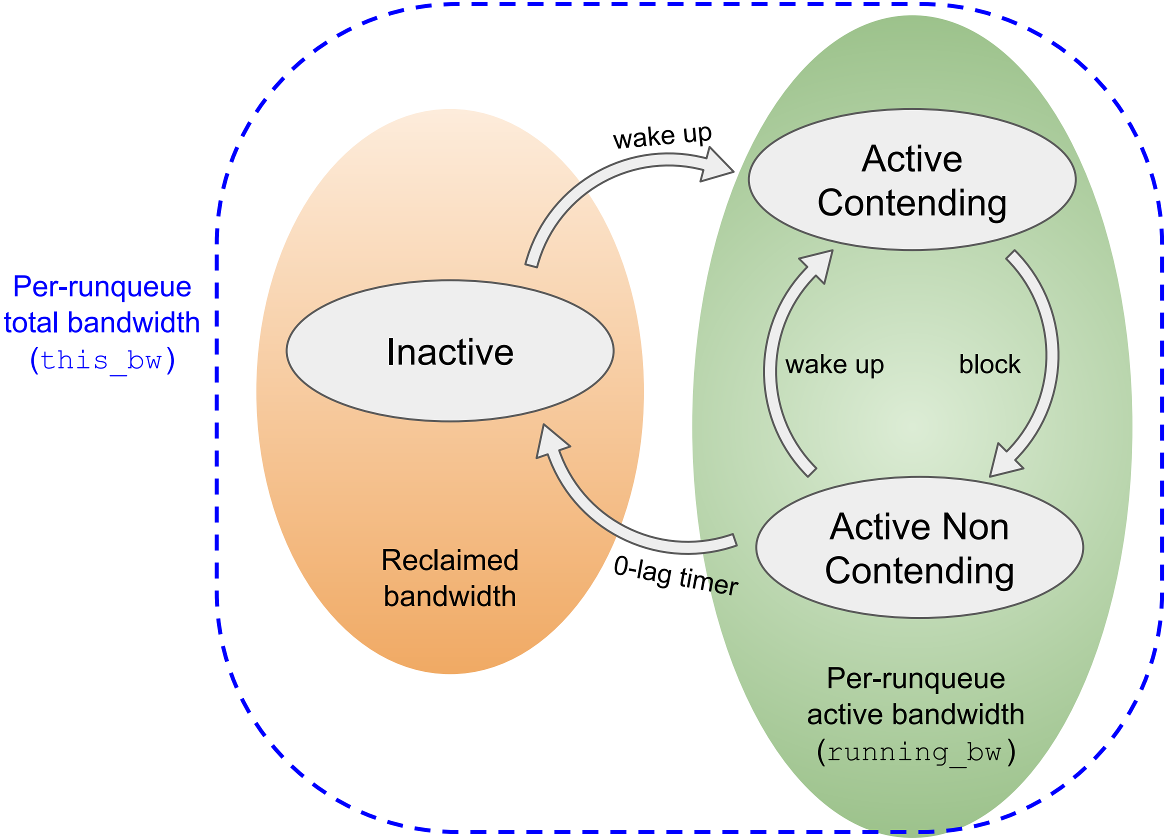
- SOLUTION
  - Bandwidth reclaiming: allow tasks to consume more than allocated
  - up to a certain maximum fraction of CPU time
  - if this doesn’t break others’ guarantees

GRUB

- Greedy Reclamation of Unused Bandwidth (GRUB[1,2])
- Replaces Constant Bandwidth Server (CBS)
- Developed by: Scuola Sant'Anna, Evidence Srl, ARM Ltd
- Mainline since v4.13
- Pretty good documentation: Documentation/scheduler/sched-deadline.txt

[1] G. Lipari, S. Baruah, Greedy reclamation of unused bandwidth in constant-bandwidth servers, 12th IEEE Euromicro Conference on Real-Time Systems, 2000.
[2] L. Abeni, J. Lelli, C. Scordino, L. Palopoli, Greedy CPU reclaiming for SCHED_DEADLINE, Real-Time Linux Workshop (RTLWS), Dusseldorf, Germany, 2014.

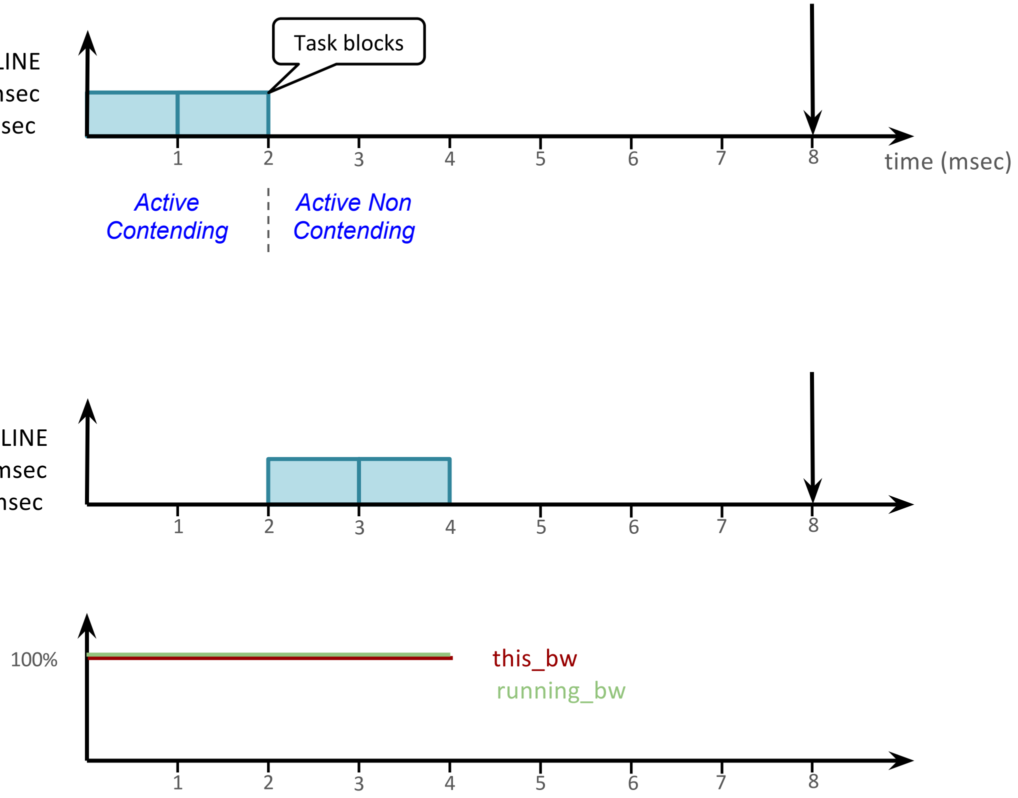
GRUB task state diagram



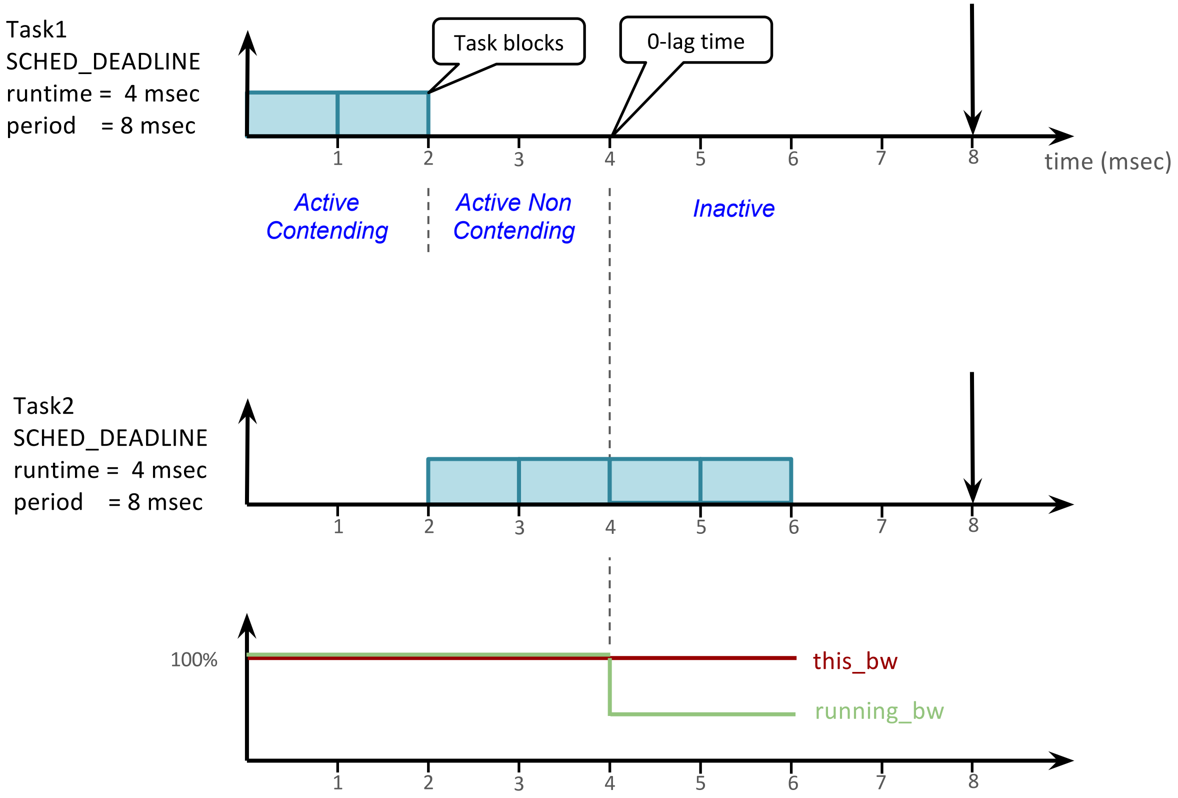
GRUB reclaiming

Task1
SCHED_DEADLINE
runtime = 4 msec
period = 8 msec

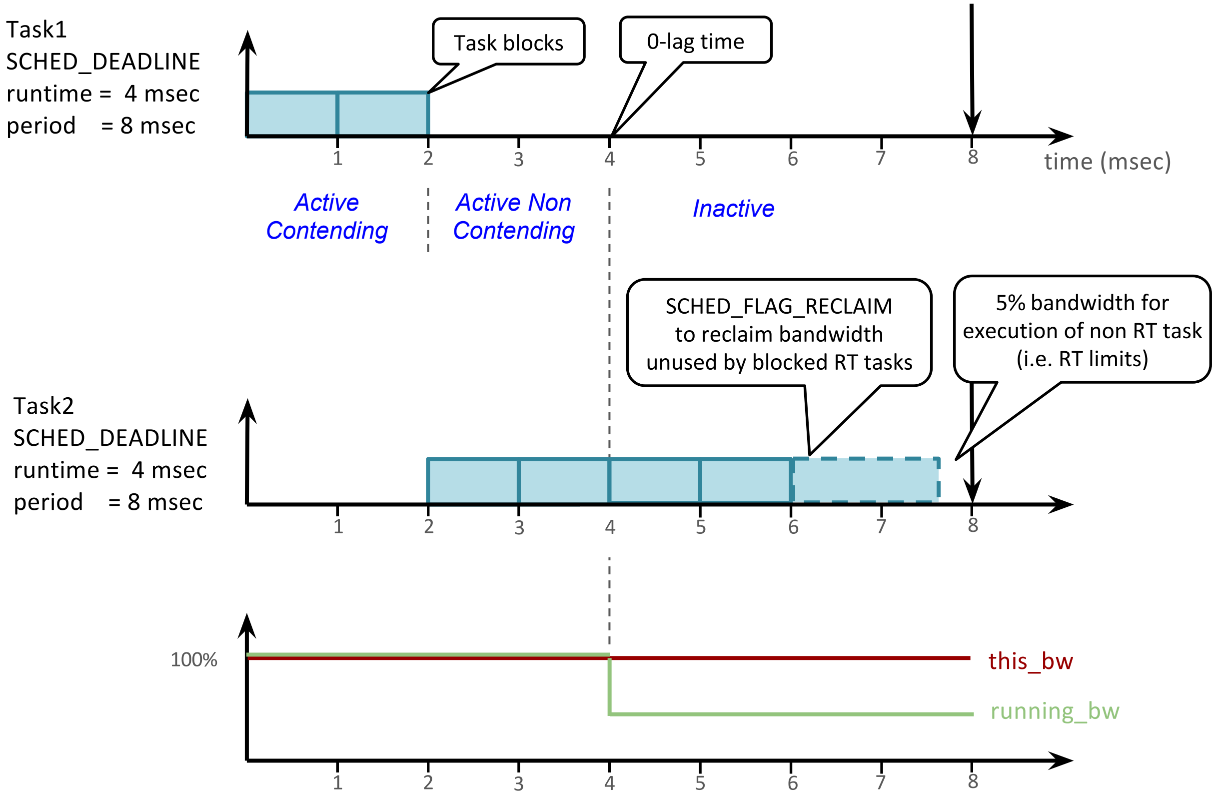
Task2
SCHED_DEADLINE
runtime = 4 msec
period = 8 msec



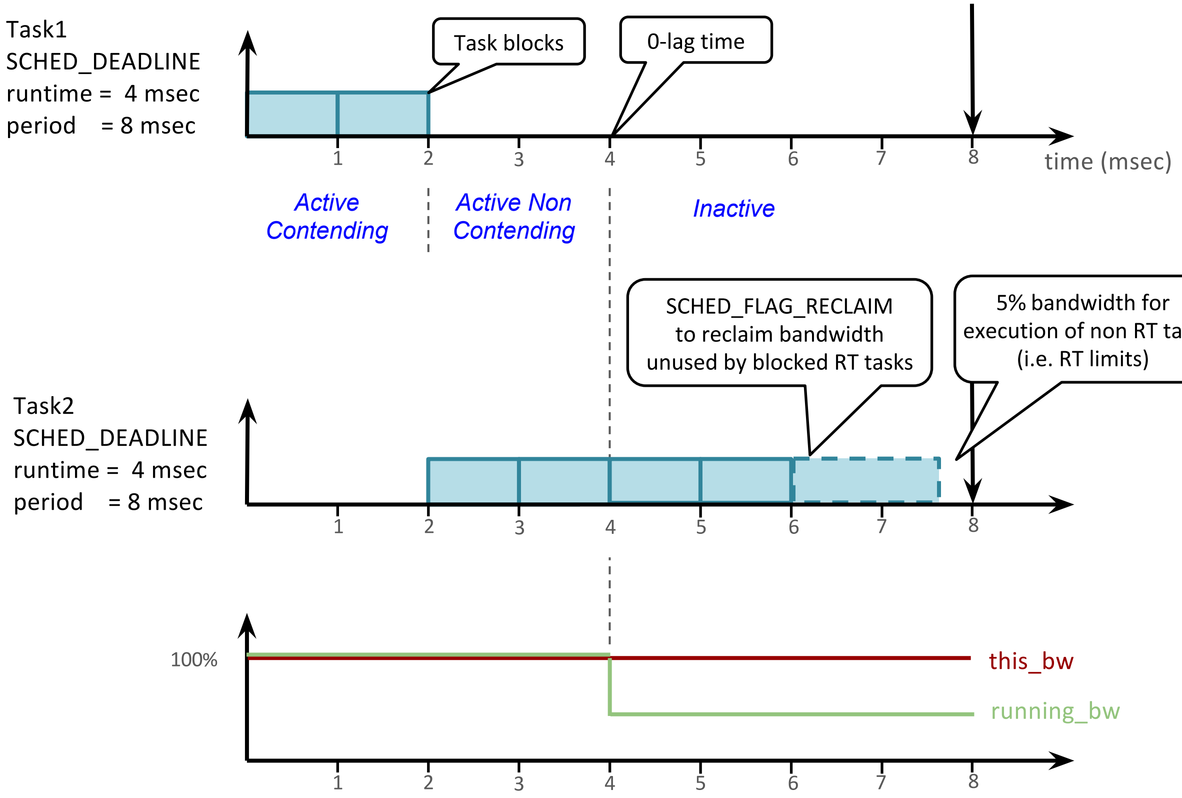
GRUB reclaiming



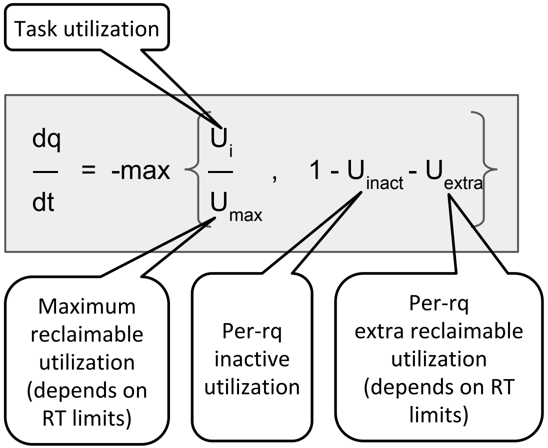
GRUB reclaiming



GRUB reclaiming

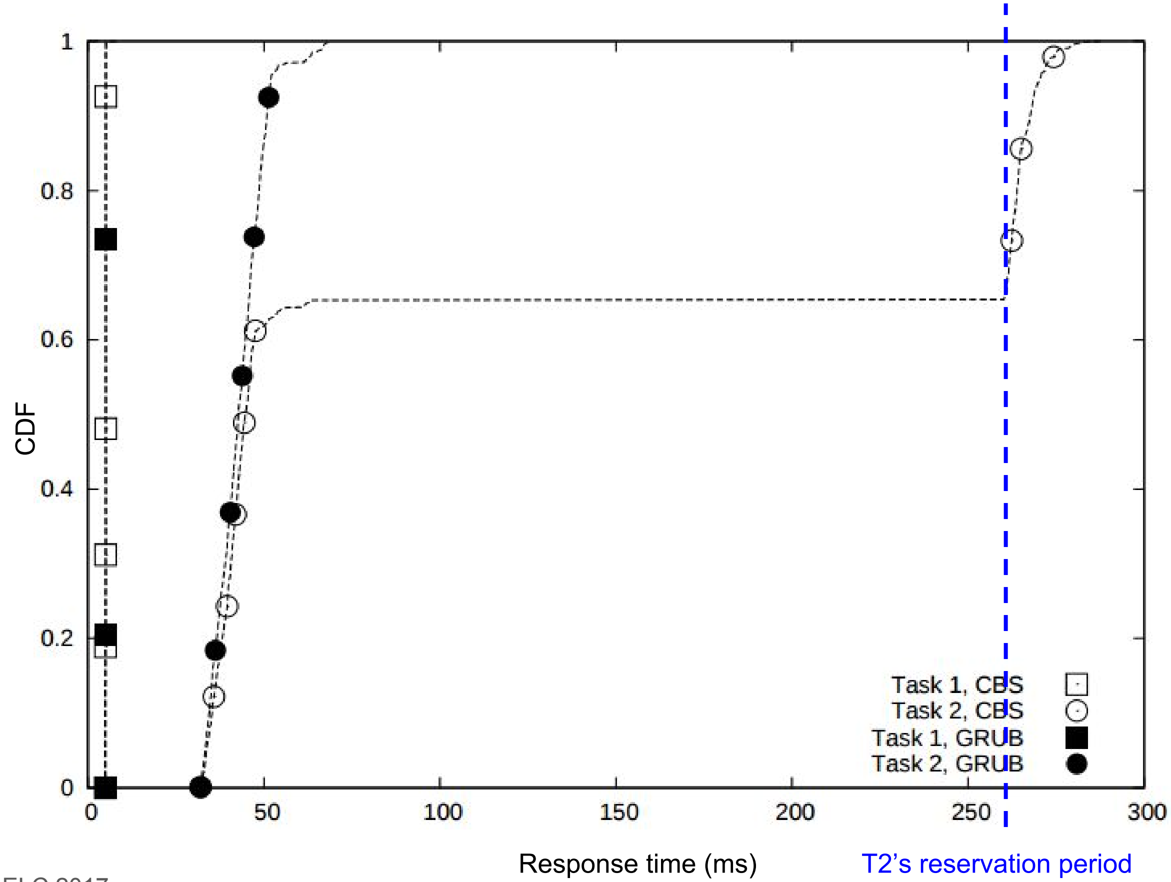


$$\frac{dq}{dt} = -\max\left\{\frac{U_i}{U_{max}}, 1 - U_{inact} - U_{extra}\right\}$$



GRUB exp. results[1]

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

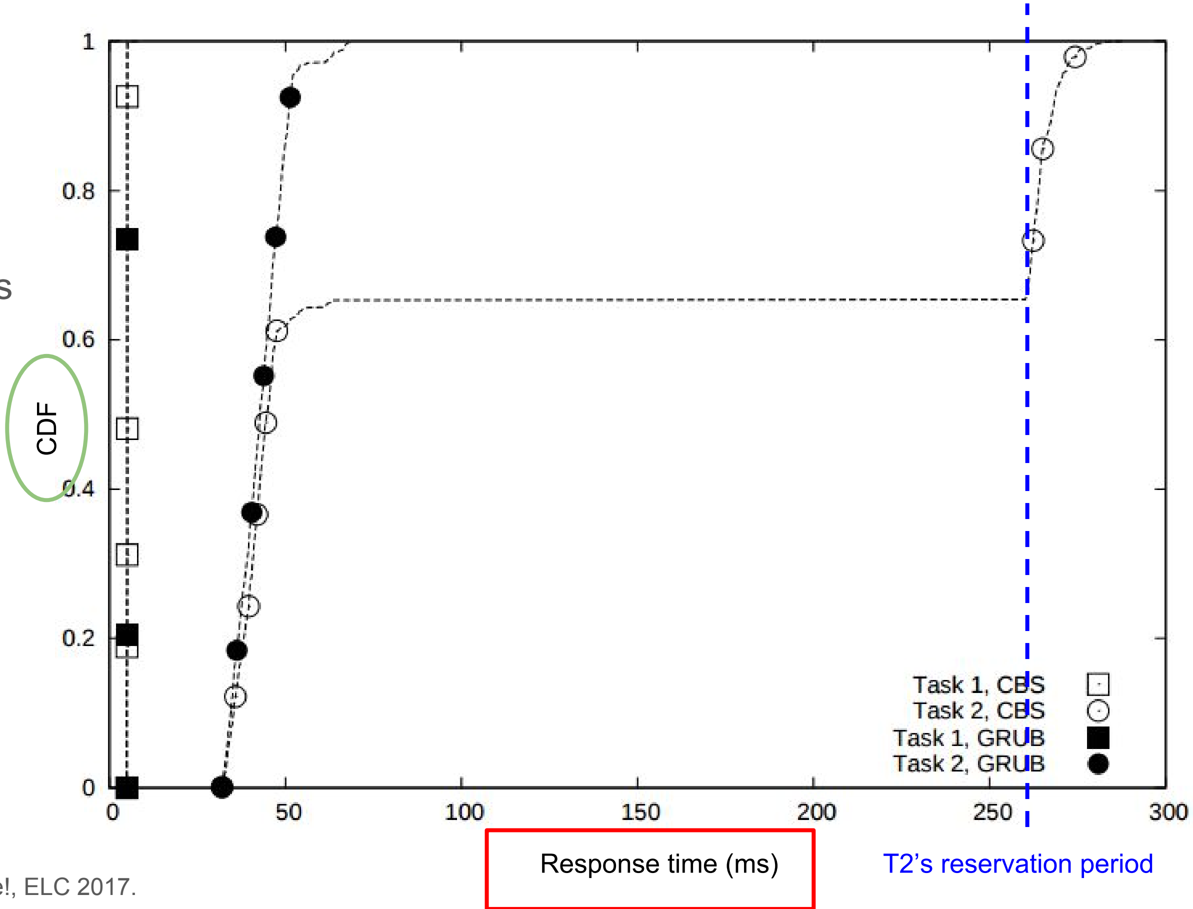


[1] Experimental results from J. Lelli, SCHED_DEADLINE: It's Alive!, ELC 2017.

GRUB exp. results[1]

- 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

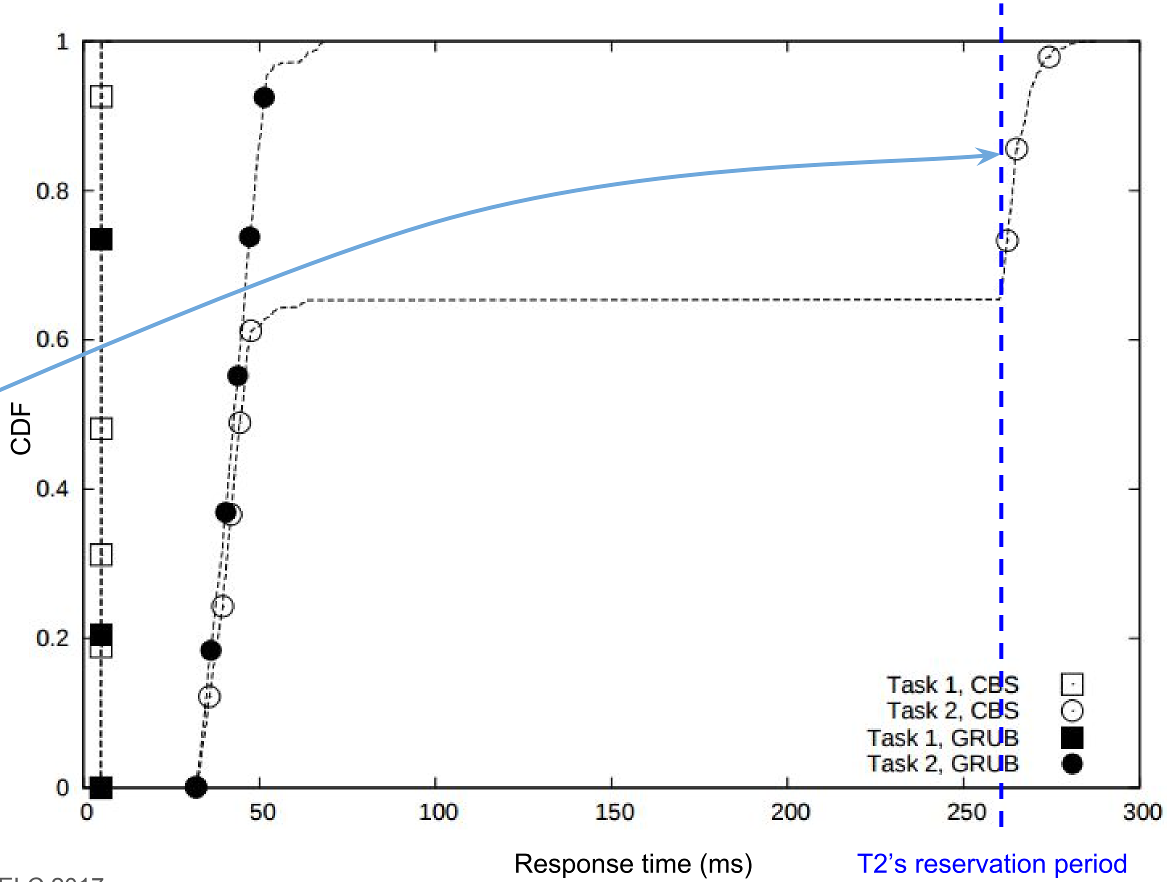


[1] Experimental results from J. Lelli, SCHED_DEADLINE: It's Alive!, ELC 2017.

GRUB exp. results[1]

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

Original CBS:
T2's response time bigger than reservation period (~25%)

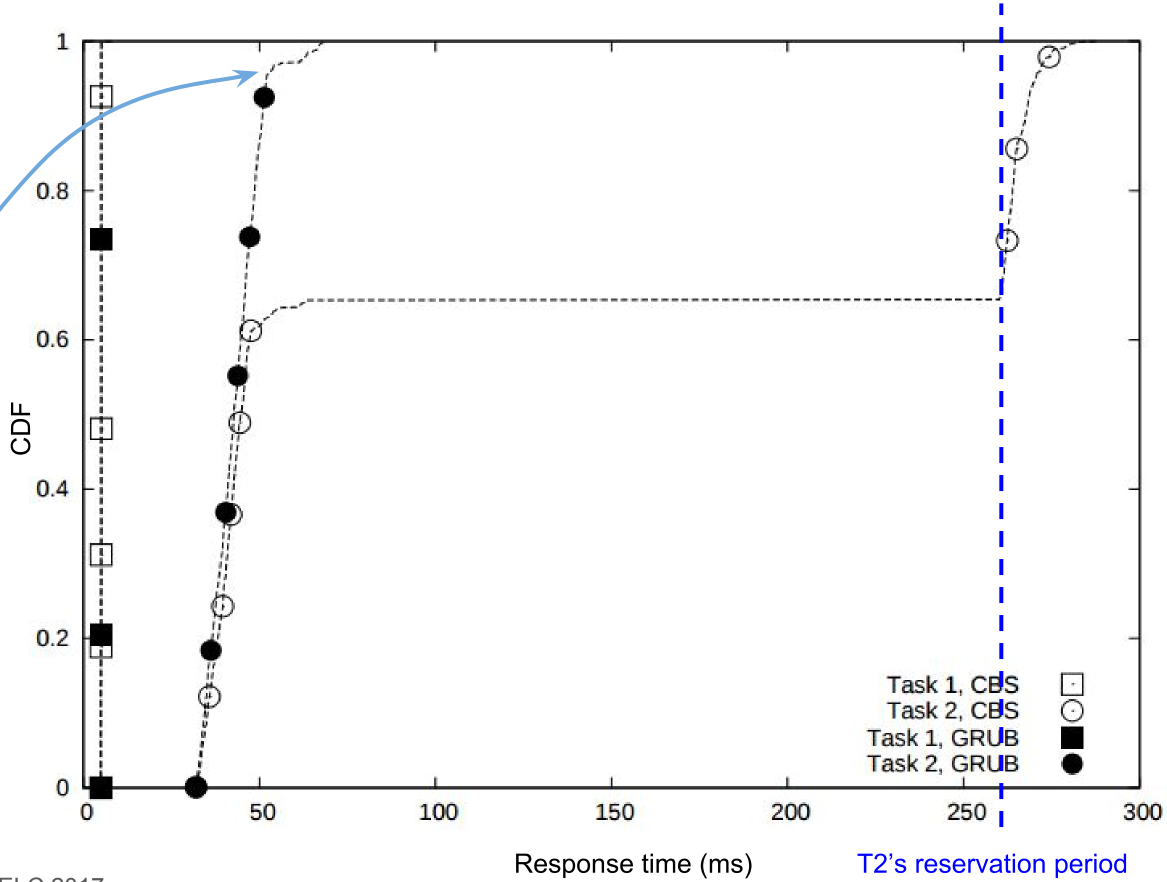


[1] Experimental results from J. Lelli, SCHED_DEADLINE: It's Alive!, ELC 2017.

GRUB exp. results[1]

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

GRUB:
T2 always completes before reservation period (using bandwidth left by T1)



[1] Experimental results from J. Lelli, SCHED_DEADLINE: It's Alive!, ELC 2017.

Schedutil integration (GRUB-PA)

Schedutil integration[1] (GRUB-PA)

- Currently, schedutil runs SCHED_DEADLINE tasks at maximum CPU frequency

- Key idea: extend schedutil to SCHED_DEADLINE tasks
  - GRUB-PA[2]: use the bandwidth reclaimed by GRUB to lower the CPU frequency
  - How: just set the CPU frequency equal to the current bandwidth
  - Reservation's runtime scaled according to frequency and CPU max capacity

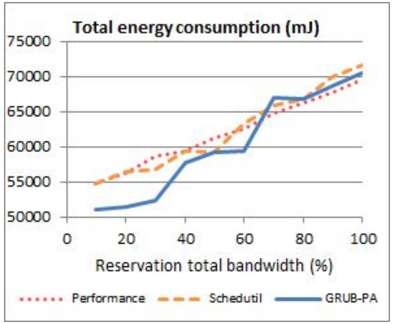
- Design choices (discussed at OSPM):
  - Use `running_bw` for frequency scaling rather than `this_bw` (more aggressive)
  - Use current CPU frequency for accounting (even if changed by other scheduling classes)
  - Set kthread to SCHED_DEADLINE with SCHED_FLAG_SPECIAL

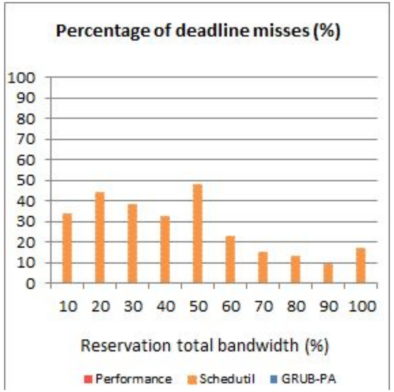
- Latest RFC sent to LKML on July 5th[3]

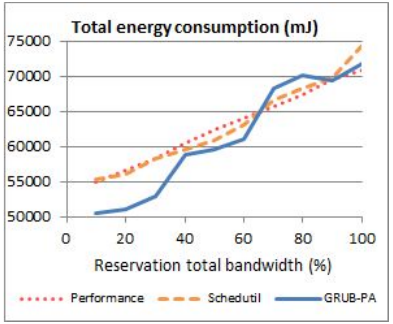
[1] Work partially supported by **ARM** and the **HERCULES** Project, funded by European Union's H2020 program under grant agreement No. 688860.
[2] C. Scordino, G. Lipari, *A Resource Reservation Algorithm for Power-Aware Scheduling of Periodic and Aperiodic Real-Time Tasks*, IEEE Transactions on Computers, 2006.
[3] https://lkml.org/lkml/2017/7/5/139

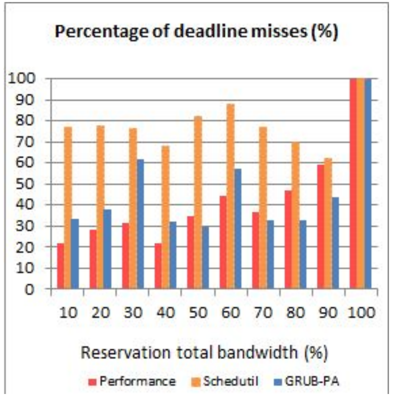
GRUB-PA vs tip on a 4-core imx6 (Cortex-A9)

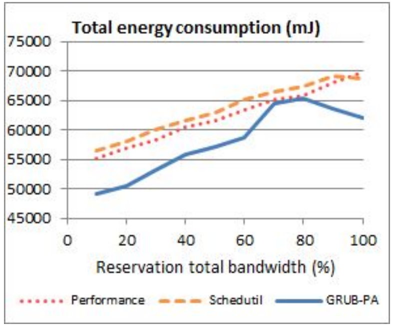
Reservation's runtime:	10 - 100 msec	10 - 100 msec	10 - 100 msec	10 - 100 msec
Reservation's period:	100 msec	100 msec	**10 msec**	100 msec
Task's runtime:	90% of reservation's runtime	**100%** of reservation's runtime	90% of reservation's runtime	90% of reservation's runtime
Task's period:	100 msec	100 msec	10 msec	100 msec
Number of tasks:	1 task	1 task	1 task	**4 tasks**

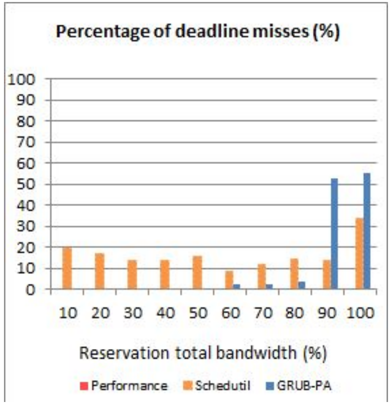


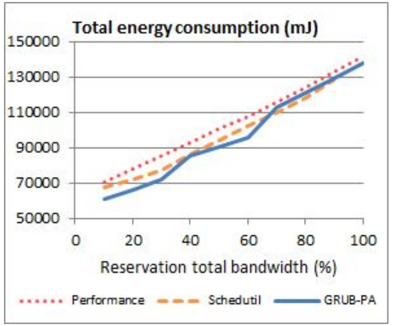


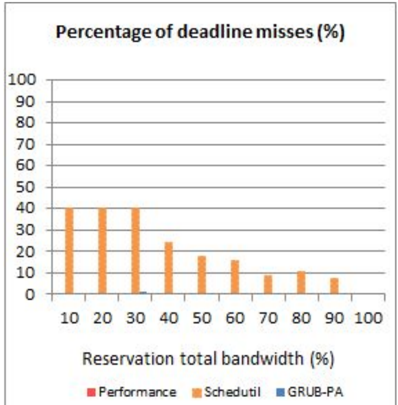




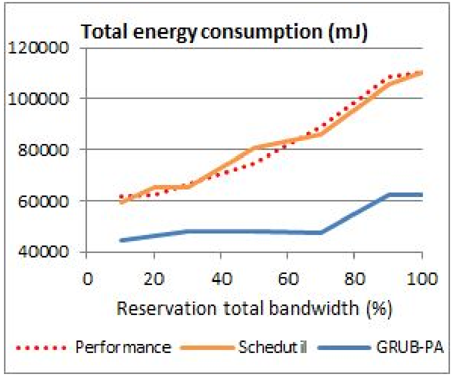


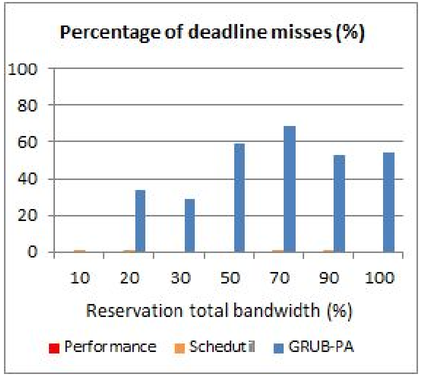






GRUB-PA: open issue





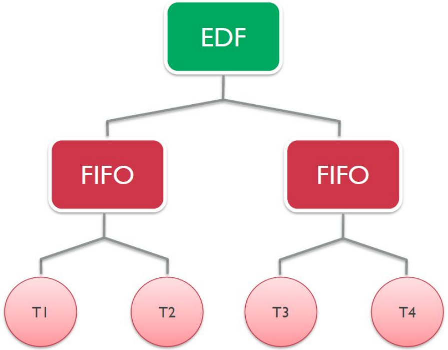
- Higher amount of deadline misses than schedutil for short periods on platforms with too long frequency switch
  - E.g. period 10 msec on Odroid XU4 (3.5 msec for a frequency switch)
- It can be mitigated by:
  - Ignoring rate_limit for urgent requests of frequency increase (by SCHED_DEADLINE)
  - Buffering an urgent request arriving when kthread is in progress
- It could be eliminated by using `this_bw` rather than `running_bw`
  - Q. Is a knob in sys/ a viable solution ?

Hierarchical/group scheduling

Hierarchical/group scheduling

- First RFC sent on LKML on March ‘17 by Scuola Sant’Anna[1]
  - Groups of tasks can be scheduled within a SCHED_DEADLINE reservation
    - First level is EDF, second level is FIFO/RR
  - Cgroup interface
  - 3 patches, quite big:
    1) removing the SCHED_RT-related cgroup mechanisms
    2) new hierarchical throttling for SCHED_RT tasks that exploits SCHED_DL
    3) RT cgroups migration of a throttled rq, seeking for available bandwidth on other CPUs

- Should eventually supplant RT throttling



[1] https://lkml.org/lkml/2017/3/31/658

Hierarchical/group scheduling

- Usage:

```
mkdir /sys/fs/cgroup/cpu/rt1
echo 100000 > /sys/fs/cgroup/cpu/rt1/cpu.rt_period_us
echo 10000 > /sys/fs/cgroup/cpu/rt1/cpu.rt_runtime_us
echo $tid1 > /sys/fs/cgroup/cpu/rt1/tasks
echo $tid2 > /sys/fs/cgroup/cpu/rt1/tasks
chrt -r -p $rtprio1 $tid1
chrt -r -p $rtprio2 $tid2
```

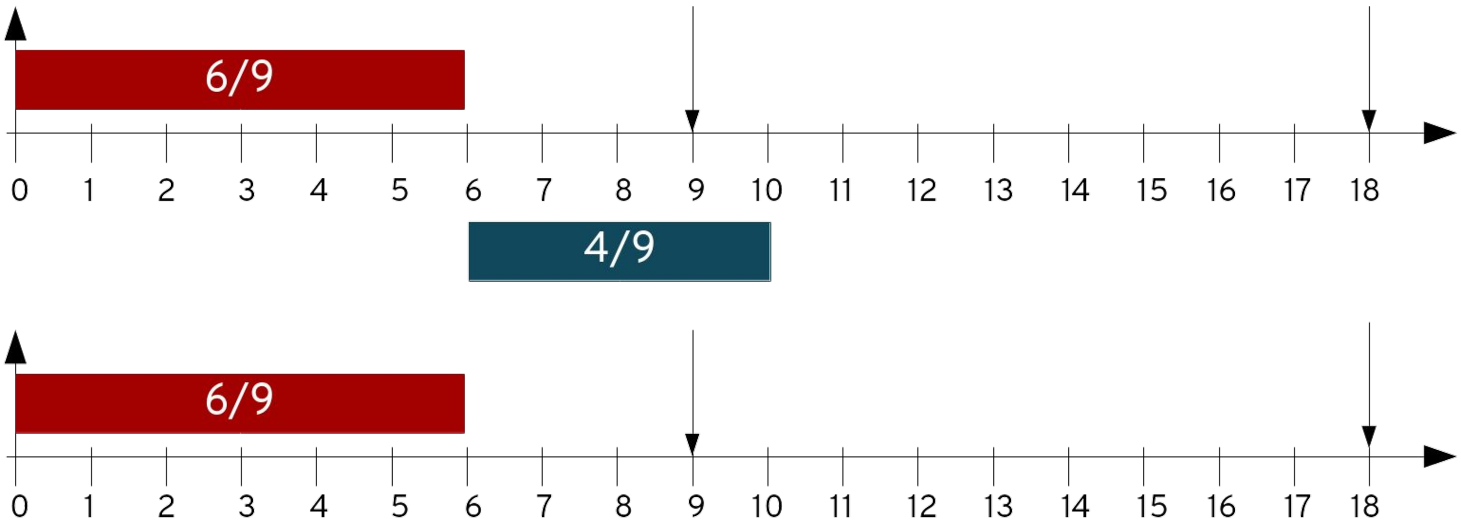
- Behavior:
  - A CPU-hog task with runtime=10ms and period(=deadline)=100ms runs for 10ms on each CPU before being throttled

- Unclear how to proceed
  - Q. Do we want a different API/behavior ?
    Or do we first want to focus on other (more urgent) features for SCHED_DEADLINE ?

Semi-partitioned scheduling

The semi-partitioned scheduler

There are some cases in which a feasible task set is not scheduled by neither global or partitioned schedulers. For instance:

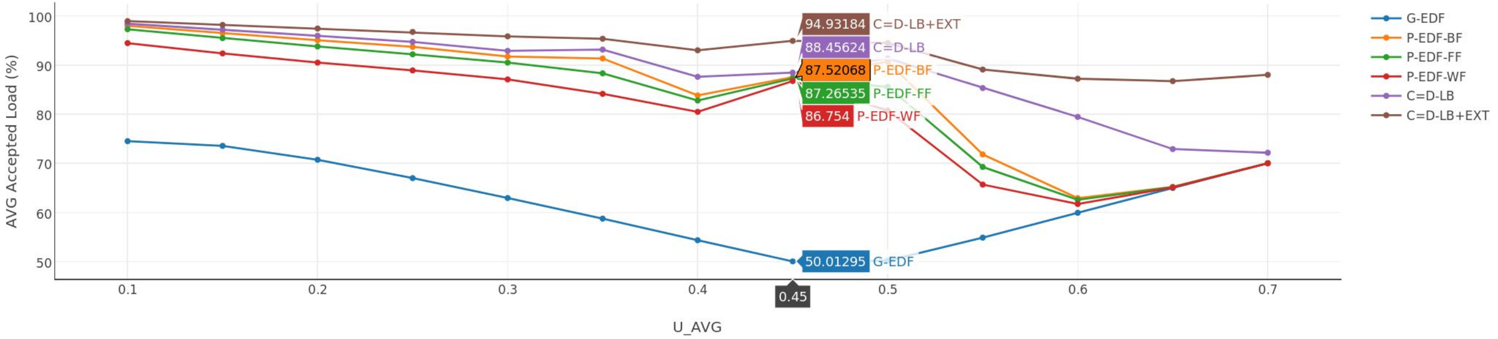


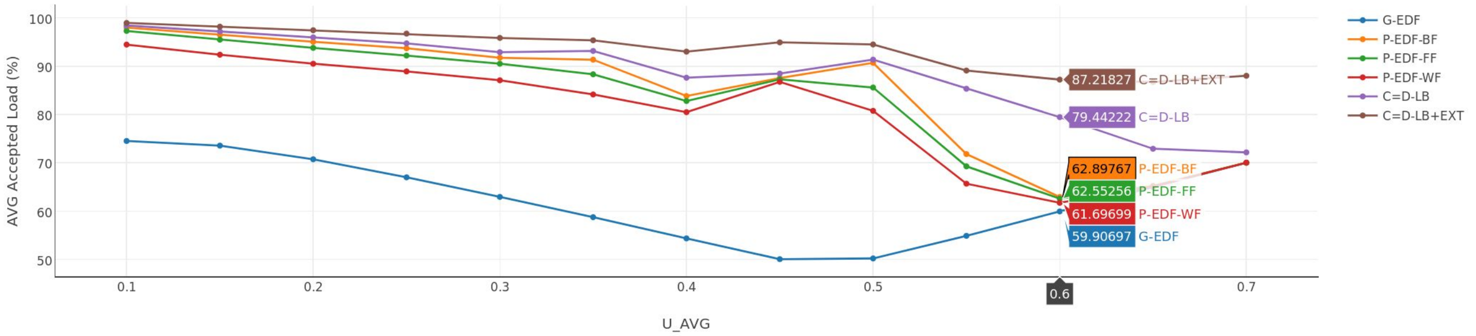
What does the academy have to say about it?

- B. Brandenburg and M. Gül, “Global Scheduling Not Required: Simple, Near-Optimal Multiprocessor Real-Time Scheduling with Semi-Partitioned Reservations” shows that:
  - “usually ≥ 99% schedulable utilization — can be achieved with simple, well-known and well-understood, low-overhead techniques (+ a few tweaks).”
  - This work, however, is not applicable for Linux because the workload is static

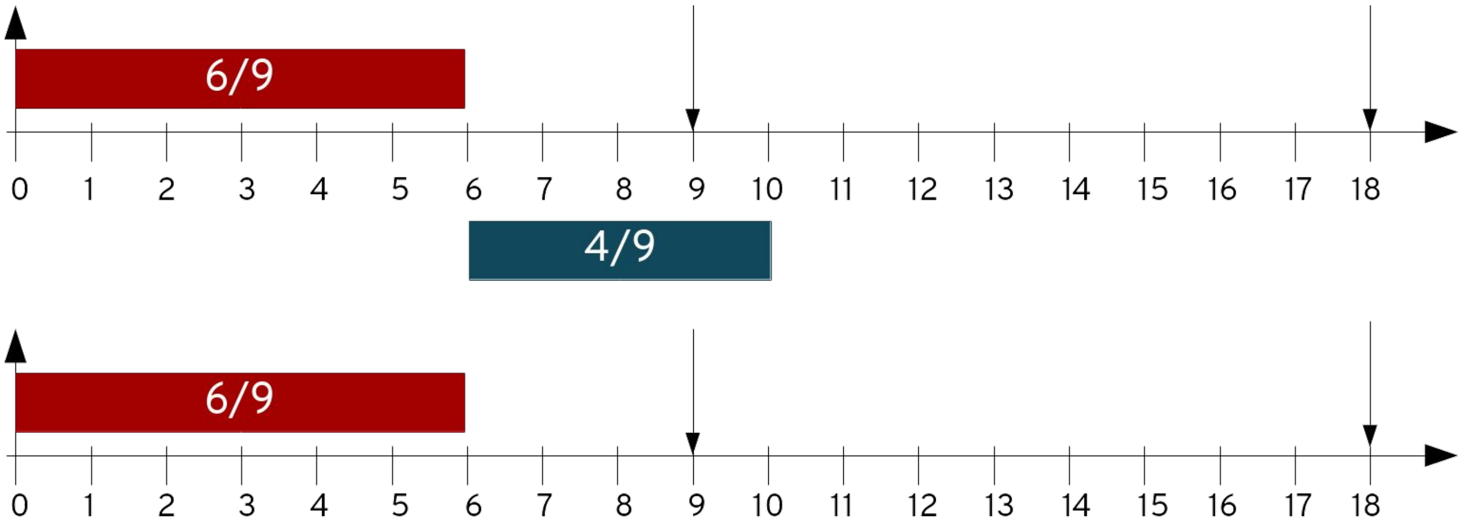
- D. Casini, A. Biondi, G. Buttazzo, “Semi-Partitioned Scheduling of Dynamic Real-Time Workload: A Practical Approach Based on Analysis-Driven Load Balancing.”
  - This paper relaxes the first, to be able to deal with dynamic workload.

How good is this online semi-partitioned scheduler?

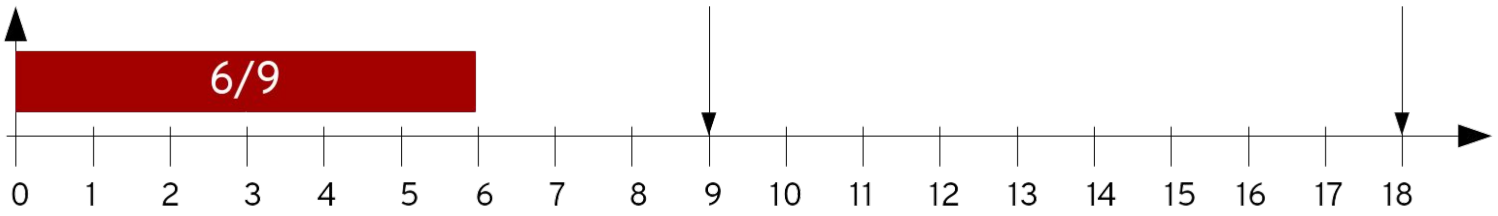


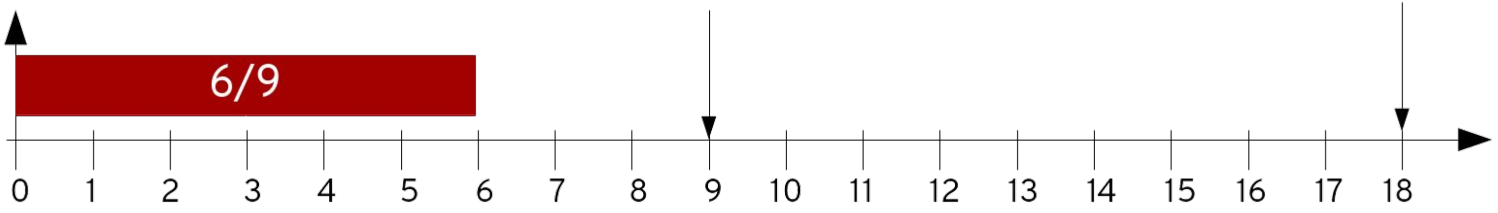


How does semi-partitioned place tasks?

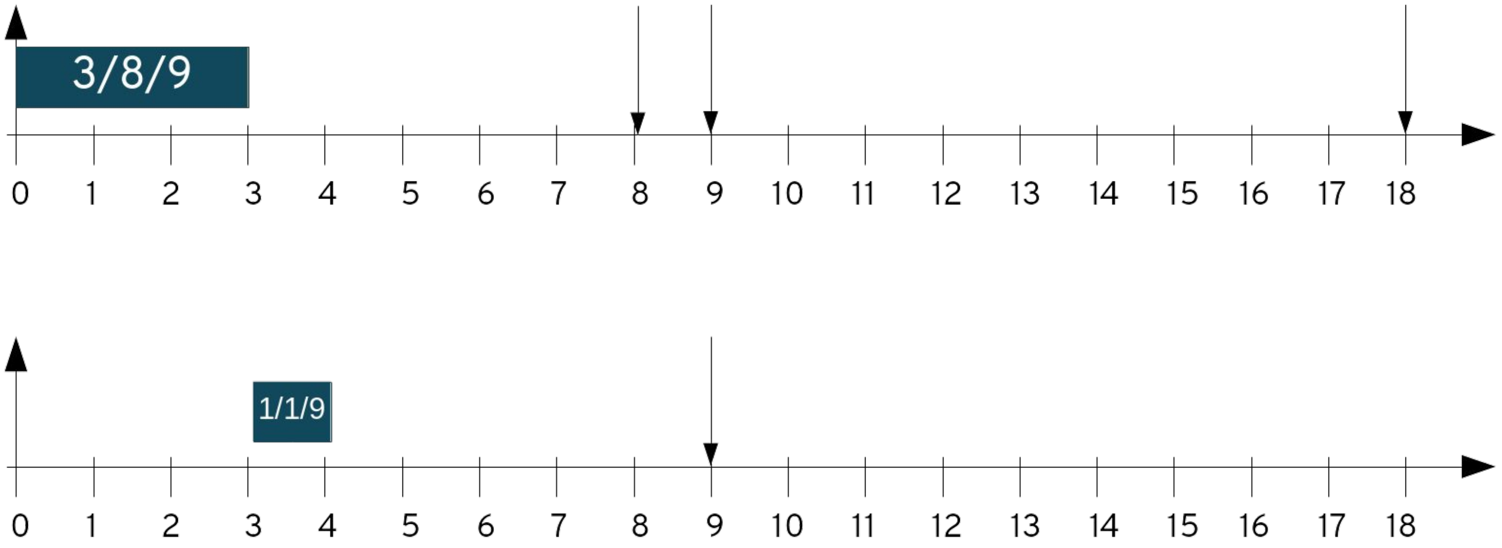


Pin as much task as possible

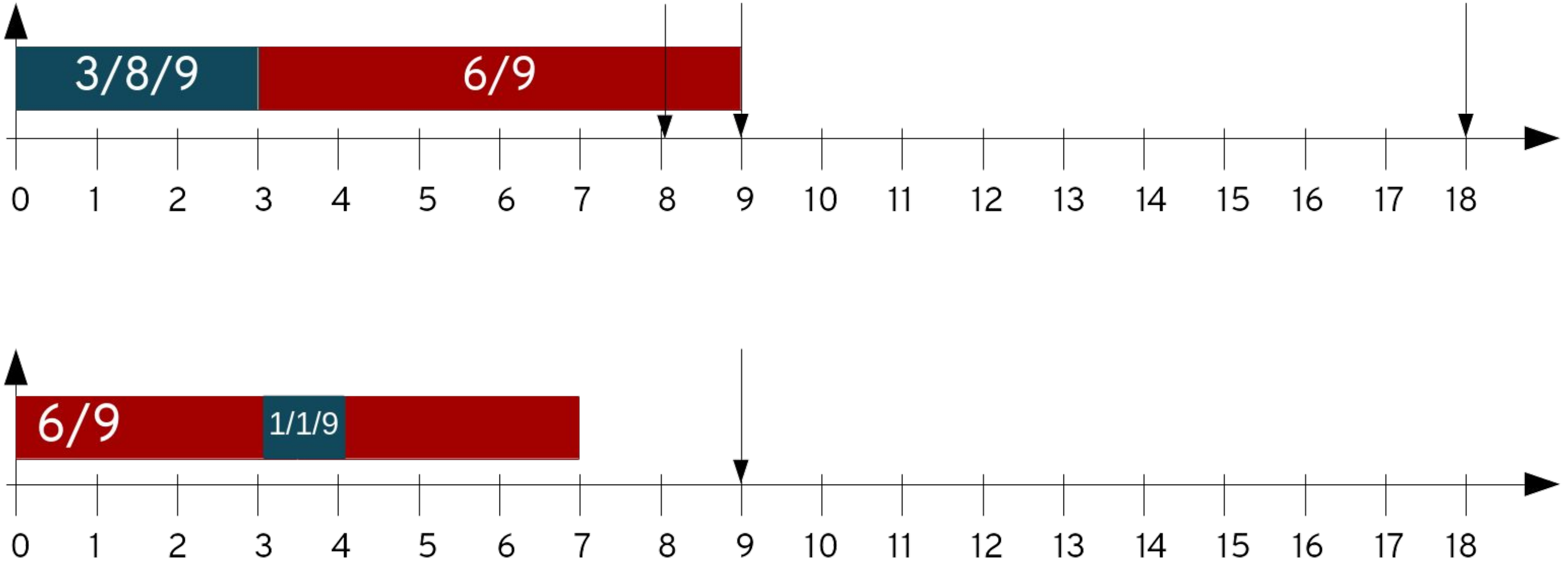




When it is not possible to pin, it splits a task.



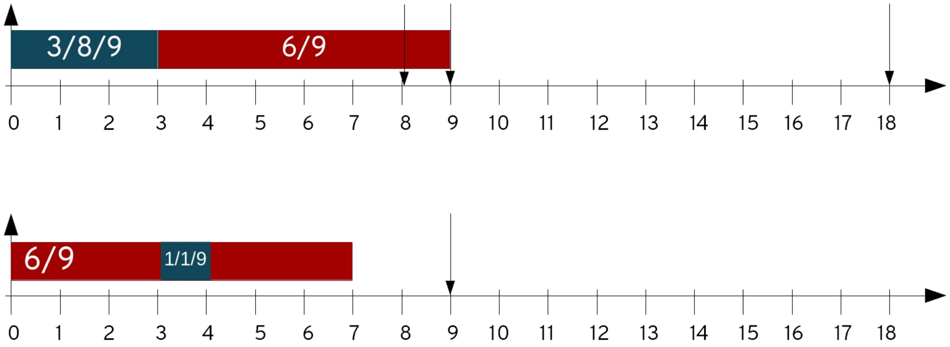
Voilà!



3/8/9
6/9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
6/9
1/1/9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Semi-partitioned scheduler development

- It changes how the deadline scheduler deals with multi-processor.
  - It is not a new scheduler, but an improvement in the Deadline scheduler
- When a task switches to the DL class…
  - The heuristics select where to put the task, and how to split it, if needed.
  - “Scheduling reservations” are assigned to the DL entity.
    - It is like if a task could have multiple DL entities.
    - Each reservation is mapped to a single CPU.
    - The scheduler schedules the reservations - not the entity.
- For example….



Semi-partitioned scheduler status

- Benefits:
  - All the RT problems are reduced to single-core!
  - The heuristics run only when setting attr/affinity/hotplug - less runtime overhead
    For instance:
    - there is no need to pull tasks, just push!
    - Migrations are bounded to M, for the system!
  - Tasks are mostly pinned to a single CPU!
  - Affinities come for FREE! YAY!
- Status of the scheduler:
  - We are seeing the theoretical results in the reality!
  - But, it stills a “WiP”, we are working in a paper about it!
- Points to be discussed:
  - The - real - admission control must to run in the kernel
  - The design of the scheduler considers implicit deadline - likewise the current… so.

Other features...

Misc

- Reclaiming by demotion
  - Requested by Android
  - Patch available on top of group scheduling
    - At the end of the budget, the task is demoted rather than migrated
  - Q. Do we want a patch independent from group scheduling (i.e. for single tasks) ?
    Or has it been superseded by GRUB ?

- Throttled signaling
  - User-level signal to inform the task about throttling
  - Patch available, easily portable on latest kernels
  - Q. Do we want/need it ?

Misc (2)

- (Single CPU) affinity
  - Currently implemented through semi-partitioned scheduling
  - Need to figure out the implications on admission control
  - Q. Do we want a patch independent from semi-partitioned scheduling ?
- Unprivileged usage
  - Executing SCHED_DEADLINE tasks w/out root privileges

BWI/Proxy execution

- First prototype of BWI implemented by Juri on an outdated kernel
  - Evidence then rebased on a newer kernel but the activity has been temporarily stopped

- We’ve heard that Peter started working on this
  - Q. Do you have some code to share with us ?
  - The group in Pisa is willing to collaborate on development/testing

Conclusions

- Schedutil integration almost ready for mainline
  - Quite good results
  - Just need to figure out how to deal with short periods (using this_bw is a viable option ?)

- The group in Pisa (Sant'Anna, Evidence) is willing to collaborate on BWI

We need a list of priorities for focusing on the most urgent features