SCHED_DEADLINE

a status update

Juri Lelli

juri.lelli@arm.com

Agenda

Presentation outline

- Deadline scheduling (AKA SCHED_DEADLINE)
  - What is it?
  - Status update
- Under discussion
  - Bandwidth reclaiming
  - Clock frequency selection hints
- Future work
  - Group scheduling
  - Dynamic feedback mechanism
  - Enhanced priority inheritance
  - Energy awareness

Agenda

Presentation outline

- Deadline scheduling (AKA SCHED_DEADLINE)
  - What is it?
  - Status update
- Under discussion
  - Bandwidth reclaiming
  - Clock frequency selection hints
- Future work
  - Group scheduling
  - Dynamic feedback mechanism
  - Enhanced priority inheritance
  - Energy awareness

SCHED_DEADLINE
What is it?

*it's **not only** about **deadlines***

- relatively new addition to the Linux scheduler

  since v3.14

- real-time scheduling policy

  higher priority than NORMAL and FIFO/RR

  only root can use it (for now ...)

- enables predictable task scheduling

  allows explicit per-task latency constraints

  avoids starvation (tasks cannot eat all available CPU time)

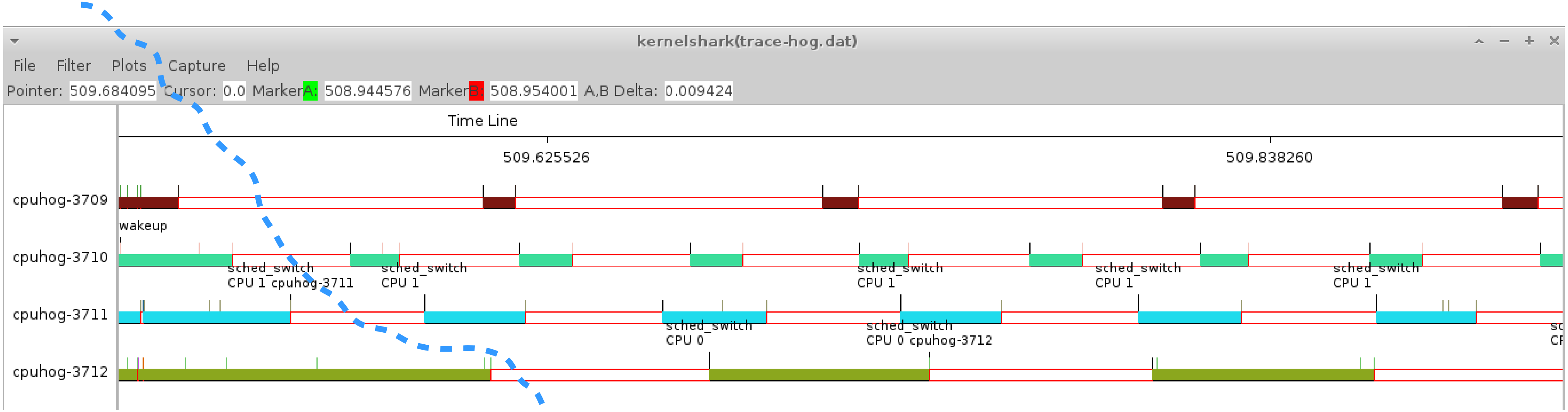
  enriches scheduler's knowledge about QoS requirements

SCHED_DEADLINE

What is it?

Predictability** and **Isolation

4 CPU-hog processes on 4 CPUs



kernelshark(trace-hog.dat)

Pointer: 509.684095 Cursor: 0.0 MarkerA: 508.944576 MarkerB: 508.954001 A,B Delta: 0.009424

SCHED_NORMAL
default Linux scheduling policy

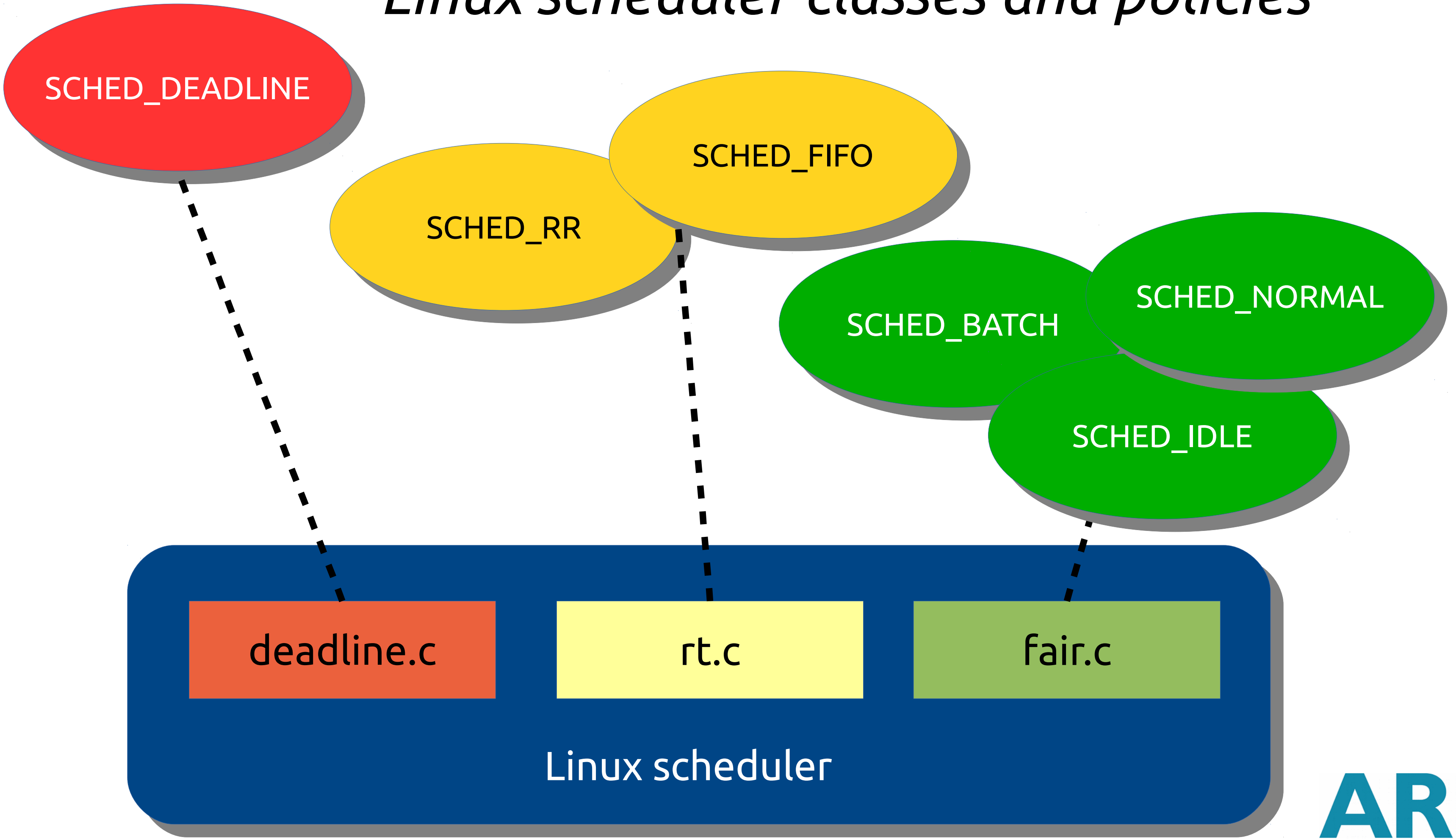
SCHED_DEADLINE

- finer-grained control over tasks scheduling
- tasks don't interfere with each other

SCHED_DEADLINE

What is it?

Linux scheduler classes and policies



SCHED_DEADLINE

EDF + CBS

it implements

- Earlies Deadline First (EDF)

  tasks with earliest deadline get executed first

- Constant Bandwidth Server (CBS)

  reservation based scheduling

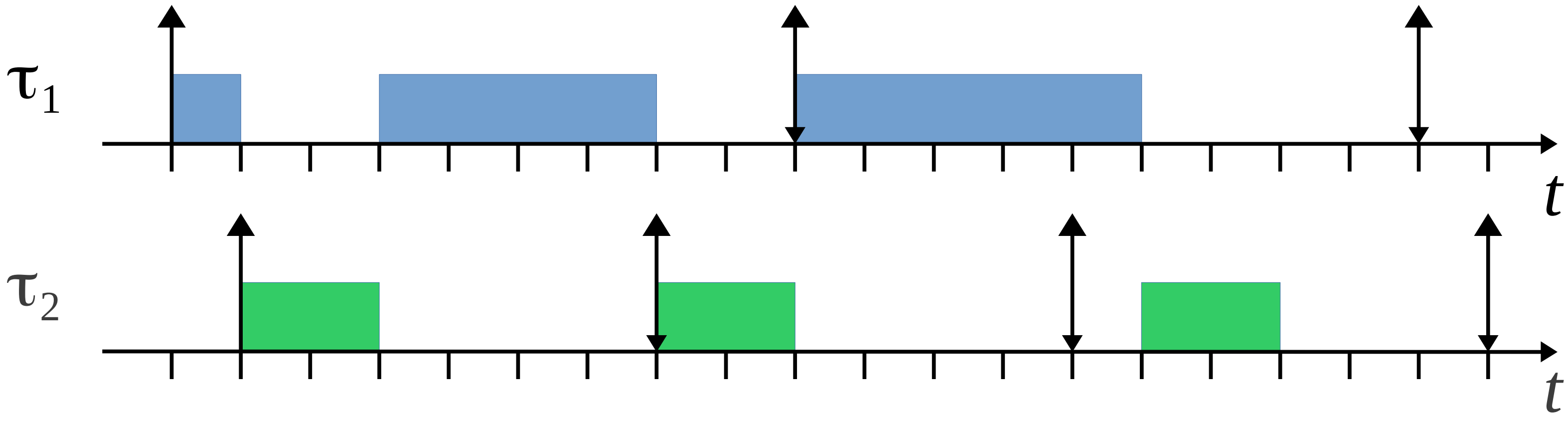
  it's the cool thing here!

SCHED_DEADLINE

EDF (plain)

τ_1 → 5 time units every 9 ~89% utilization

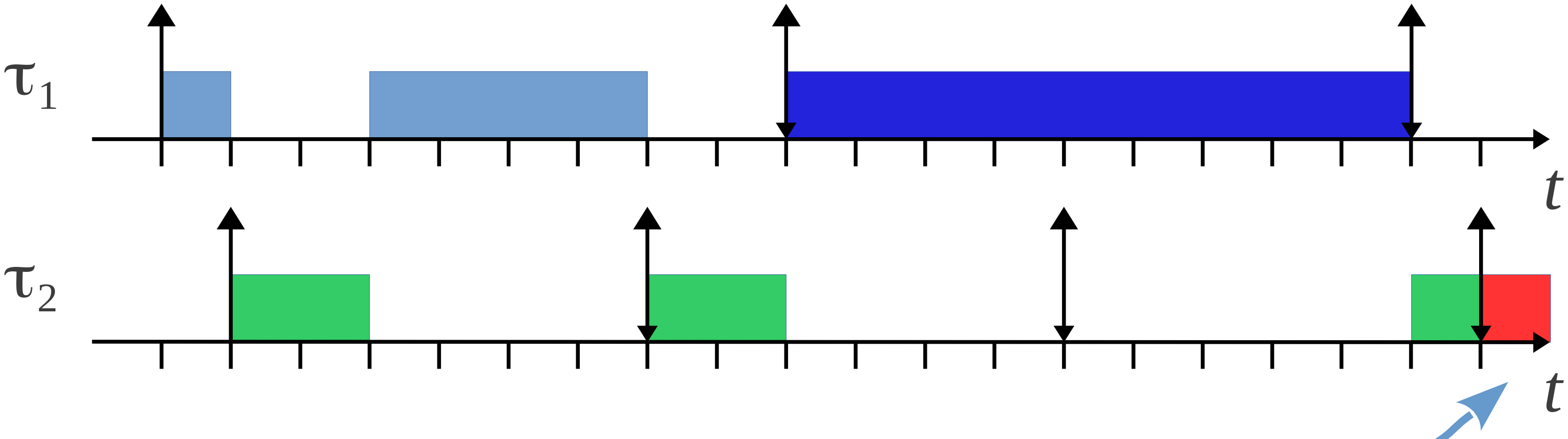
τ_2 → 2 time units every 6



SCHED_DEADLINE

EDF (plain: problems)

τ_1 → second job behaves bad



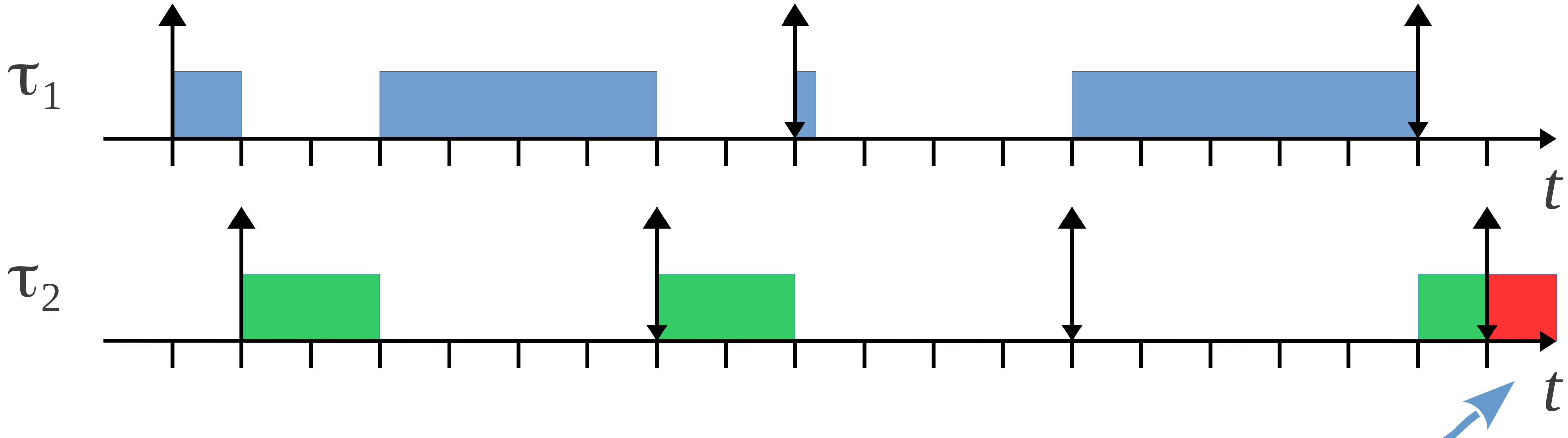
τ_1 causes a deadline miss on τ_2

SCHED_DEADLINE

EDF (plain: problems)

τ_1 → blocks just after the second activation

τ_1 → resumes with the third instance of τ_2



τ_1 causes a deadline miss on τ_2

SCHED_DEADLINE

Constant Bandwidth Server (and EDF)

- **resource** (CPU) **reservation** mechanism

  a task is allowed to execute for

  Q time units *(runtime)*

  in every interval of length P *(period)*

- CBS computes reservation's **dynamic deadlines**

  slowing down or throttling misbehaving tasks

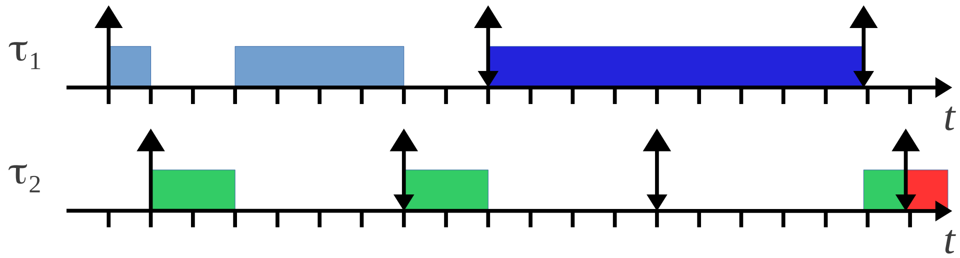
- EDF gives higher priority to more urgent reservations
- EDF + CBS provides **temporal isolation**

SCHED_DEADLINE

EDF + CBS

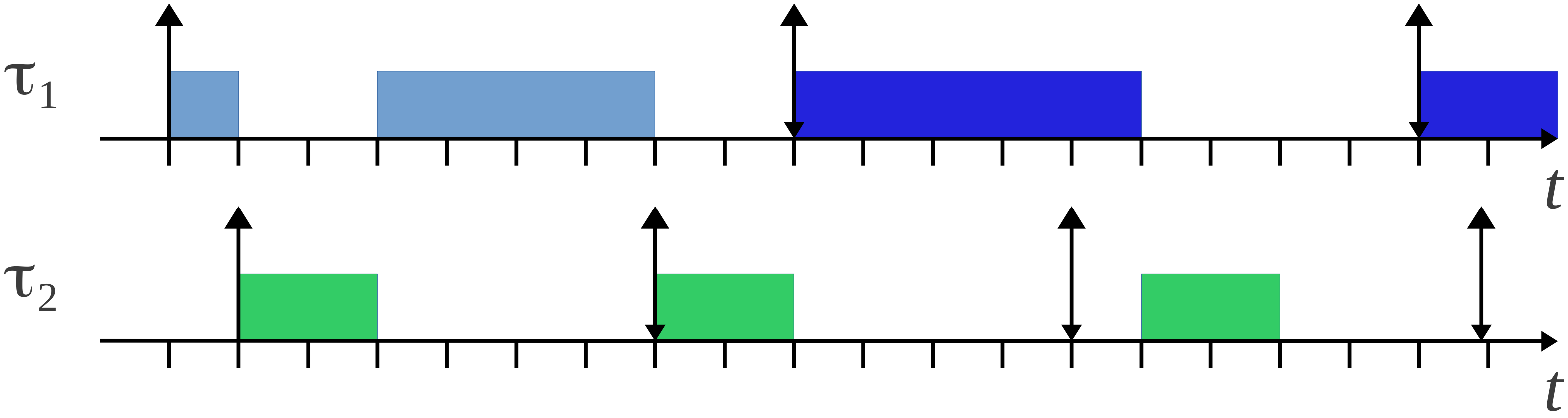
plain EDF

(bad task) →



τ_1 → second job behaves bad

τ_1 → once budget exhausted, delay until next period

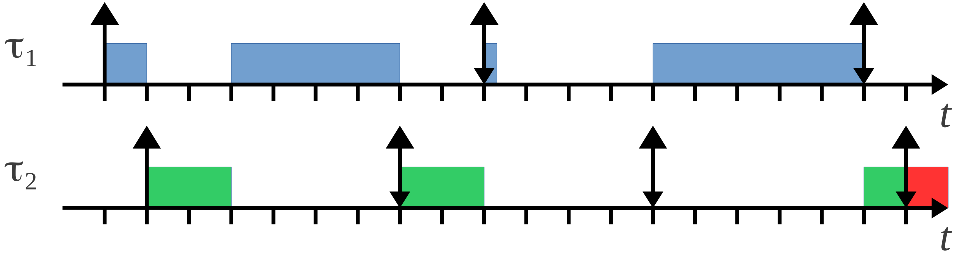


SCHED_DEADLINE

EDF + CBS

plain EDF

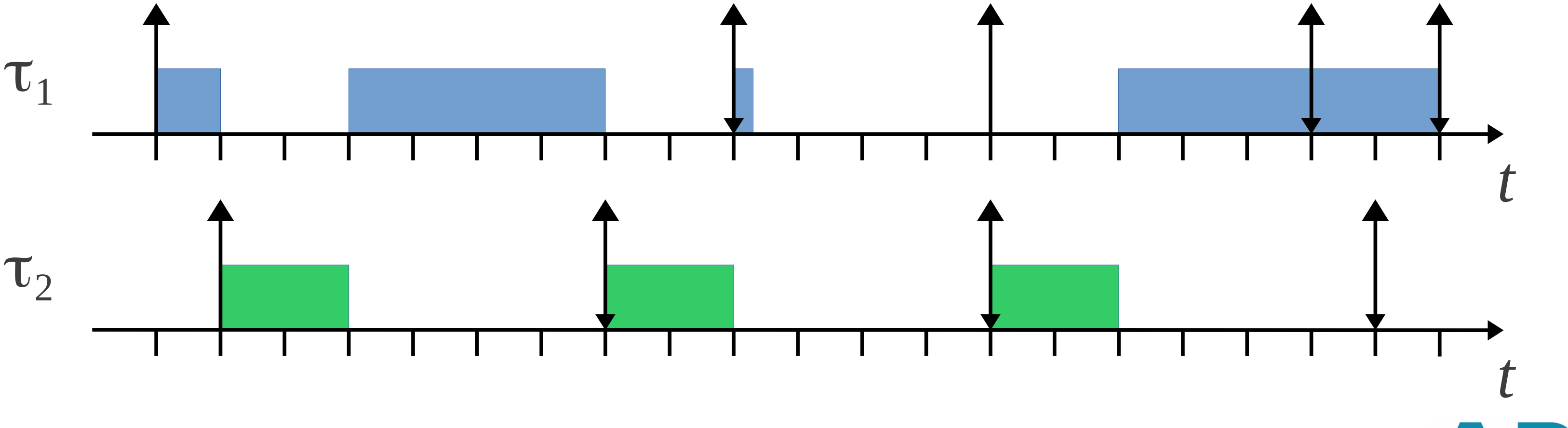
(block/unblock) →



τ_1 → blocks just after the second activation

τ_1 → resumes with the third instance of τ_2

CBS "unblock rule" applied



SCHED_DEADLINE

Load Balancing and Inheritance (and a question)

- active load balancing (push/pull)

  like for SCHED_FIFO

  global EDF: on an M-CPUs system the M earliest DL ready tasks are always running (respecting affinity/cpusets)

- deadline inheritance

  boosted task inherits deadline of the donor

  suboptimal solution… see future work

- common question: does it work with PREEMPT_RT ?

  it's orthogonal to it

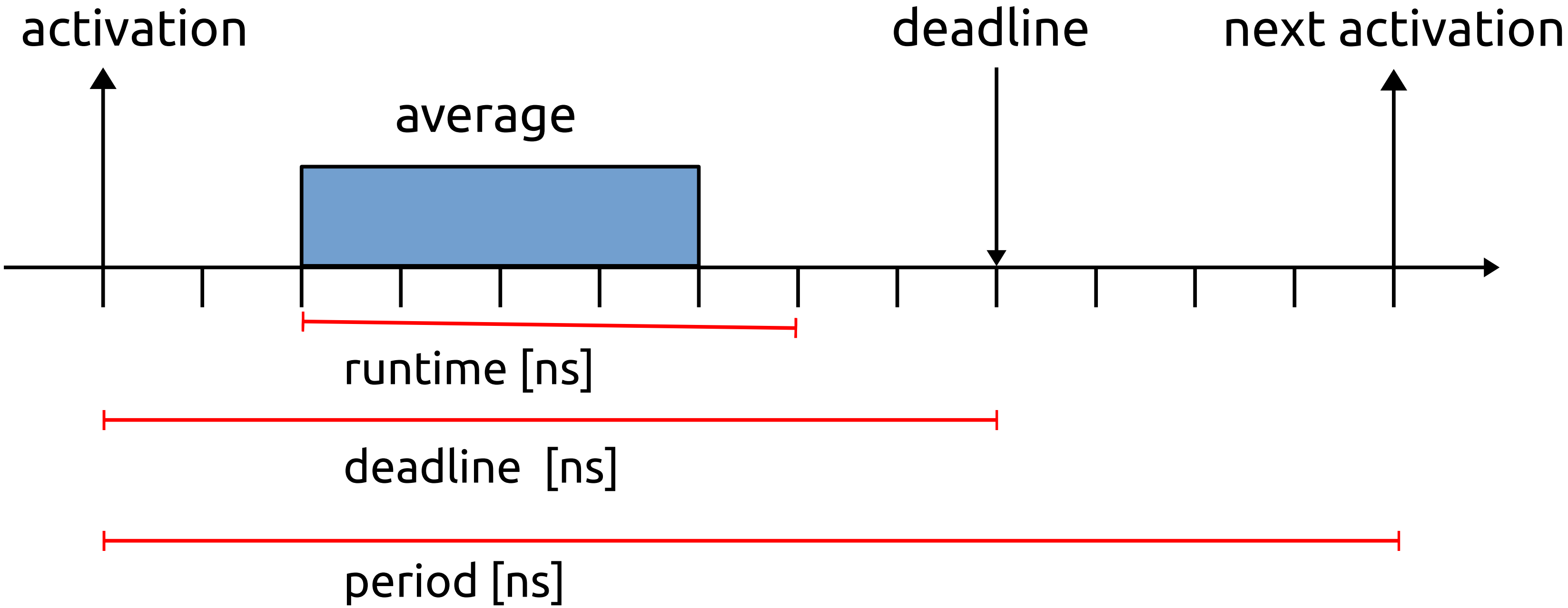
  PREEMPT_RT reduces latencies, SCHED_DEADLINE implements a scheduling algorithm (can benefit from the former)

  they *should* work together without any problem :-)

SCHED_DEADLINE

how to setup params

simple rule of thumb



SCHED_DEADLINE

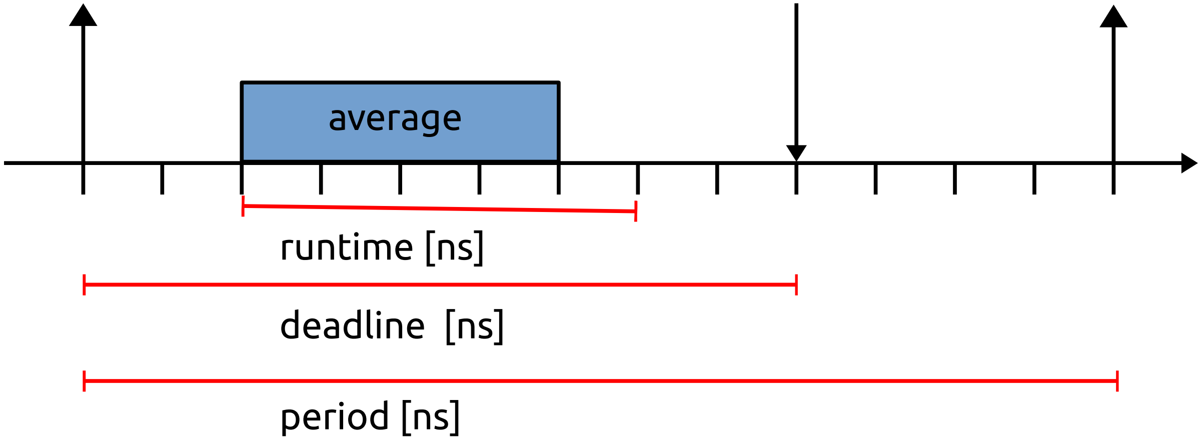
API

```
struct sched_attr {
    u32 size;
    u32 sched_policy;
    u64 sched_flags;

    /* SCHED_NORMAL, SCHED_BATCH */
    s32 sched_nice;

    /* SCHED_FIFO, SCHED_RR */
    u32 sched_priority;

    /* SCHED_DEADLINE */
    u64 sched_runtime;
    u64 sched_deadline;
    u64 sched_period;
};
```



```
int sched_setattr(pid_t pid, const struct sched_attr *attr, unsigned int flags);

int sched_getattr(pid_t pid, const struct sched_attr *attr, unsigned int size, unsigned int flags);
```

SCHED_DEADLINE

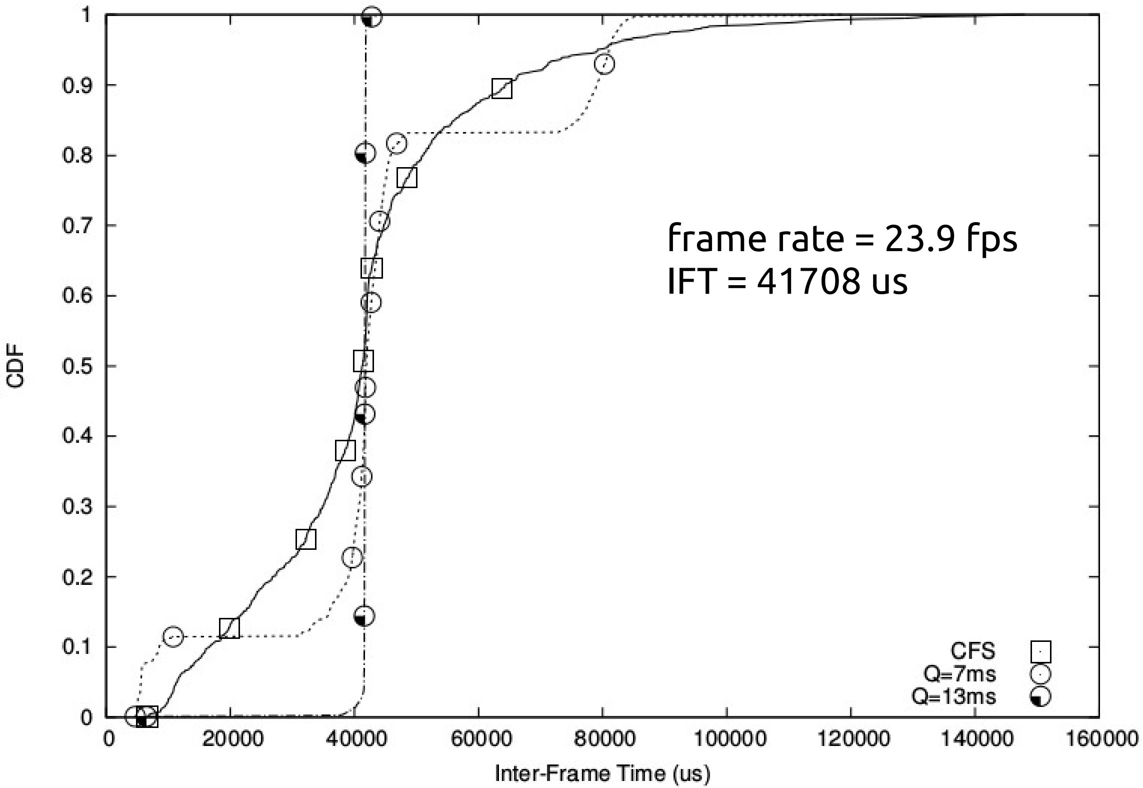
Example of usage

```
#include <sched.h>
...
struct sched_attr attr;
attr.size = sizeof(struct attr);
attr.sched_policy = SCHED_DEADLINE;
attr.sched_runtime = 30000000;
attr.sched_period = 100000000;
attr.sched_deadline = attr.sched_period;
...
if (sched_setattr(gettid(), &attr, 0))
    perror("sched_setattr()");
...
```

SCHED_DEADLINE
numbers*

- mplayer HD movie
- QoS is inter-frame time (IFT)

  curr_dt – prev_dt
- Variation in IFT is bad
- 6 other instances of mplayer in background



- cumulative distribution function (CDF)

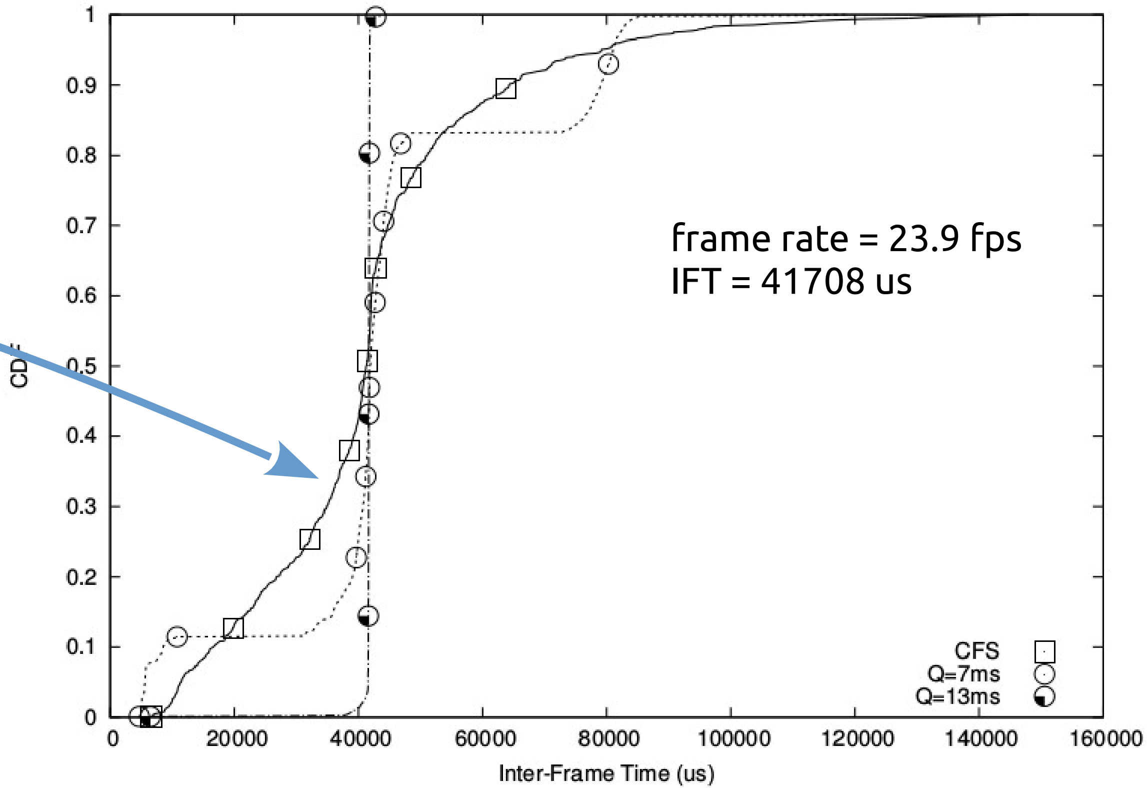
  vertical line at expected IFT gives best result

* Juri Lelli, Claudio Scordino, Luca Abeni, Dario Faggioli, Deadline scheduling in the Linux kernel, Software: Practice and Experience 2015 http://onlinelibrary.wiley.com/doi/10.1002/spe.2335/abstract

SCHED_DEADLINE

numbers*

SCHED_NORMAL (CFS)
QoS highly dependent on
system load



frame rate = 23.9 fps
IFT = 41708 us

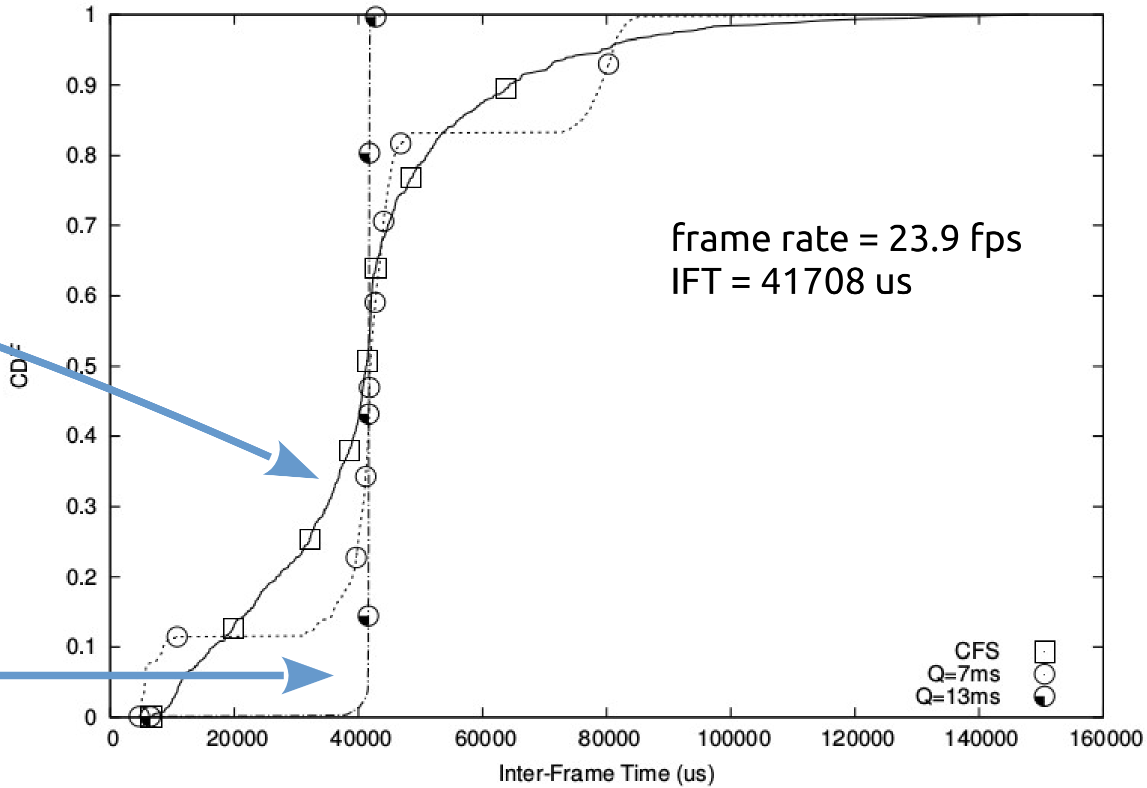
* Juri Lelli, Claudio Scordino, Luca Abeni, Dario Faggioli, Deadline scheduling in the Linux kernel, Software: Practice and Experience 2015
http://onlinelibrary.wiley.com/doi/10.1002/spe.2335/abstract

SCHED_DEADLINE

numbers*

SCHED_NORMAL (CFS)
QoS highly dependent on
system load

SCHED_DEADLINE
player not affected
(period = IFT , runtime = 13ms)



frame rate = 23.9 fps
IFT = 41708 us

* Juri Lelli, Claudio Scordino, Luca Abeni, Dario Faggioli, Deadline scheduling in the Linux kernel, Software: Practice and Experience 2015
http://onlinelibrary.wiley.com/doi/10.1002/spe.2335/abstract

Agenda

Presentation outline

- Deadline scheduling (AKA SCHED_DEADLINE)
  - What is it?
  - Status update
- Under discussion
  - Bandwidth reclaiming
  - Clock frequency selection hints
- Future work
  - Group scheduling
  - Dynamic feedback mechanism
  - Enhanced priority inheritance
  - Energy awareness

Bandwidth reclaiming

under discussion*

- tasks' bandwidth is fixed

  can only be changed with syscall

- what if tasks occasionally need more bandwidth ?

  occasional workload fluctuations (e.g., network traffic, rendering particularly heavy frame)

- reclaiming: allow tasks to consume more than allocated

  up to a certain maximum fraction of CPU time

  if this doesn't break others' guarantees

- implementation details

  greedy reclaiming of unused bandwidth (GRUB)

  Luca Abeni (University of Trento) driving this

* CPU reclaiming for SCHED_DEADLINE
https://lwn.net/Articles/671929/

Bandwidth reclaiming

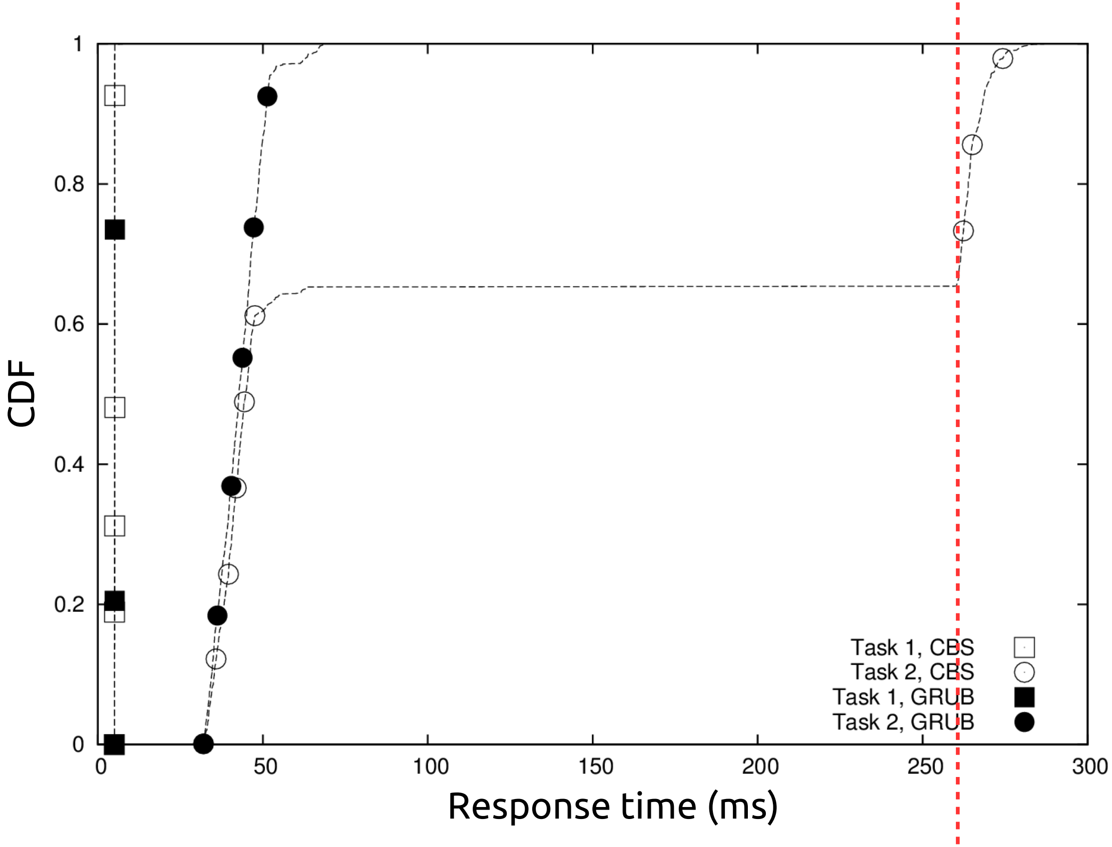
results*

- Task1 (6ms, 20ms)

  constant execution time (5ms)

- Task2 (45ms, 260ms)

  experiences occasional variances (35-52ms)



T2 reservation period

* Luca Abeni, Juri Lelli, Claudio Scordino, Luigi Palopoli, Greedy CPU reclaiming for SCHED_DEADLINE, RTLWS14 http://disi.unitn.it/~abeni/reclaiming/rtlws14-grub.pdf

Bandwidth reclaiming

results*

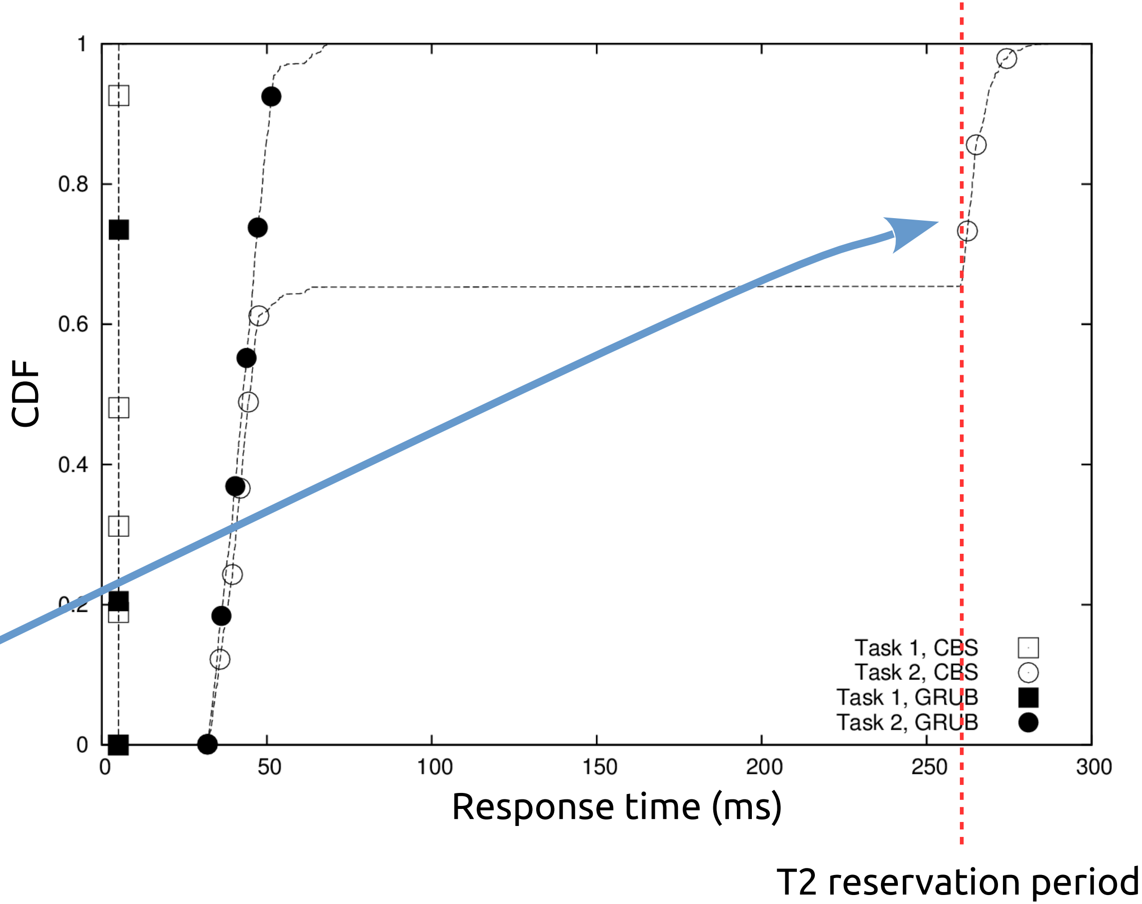
- Task1 (6ms, 20ms)

  constant execution time (5ms)

- Task2 (45ms, 260ms)

  experiences occasional variances (35-52ms)

Plain CBS
T2 response time bigger than reservation period (~25%)



* Luca Abeni, Juri Lelli, Claudio Scordino, Luigi Palopoli, Greedy CPU reclaiming for SCHED_DEADLINE, RTLWS14
http://disi.unitn.it/~abeni/reclaiming/rtlws14-grub.pdf

Bandwidth reclaiming

results*

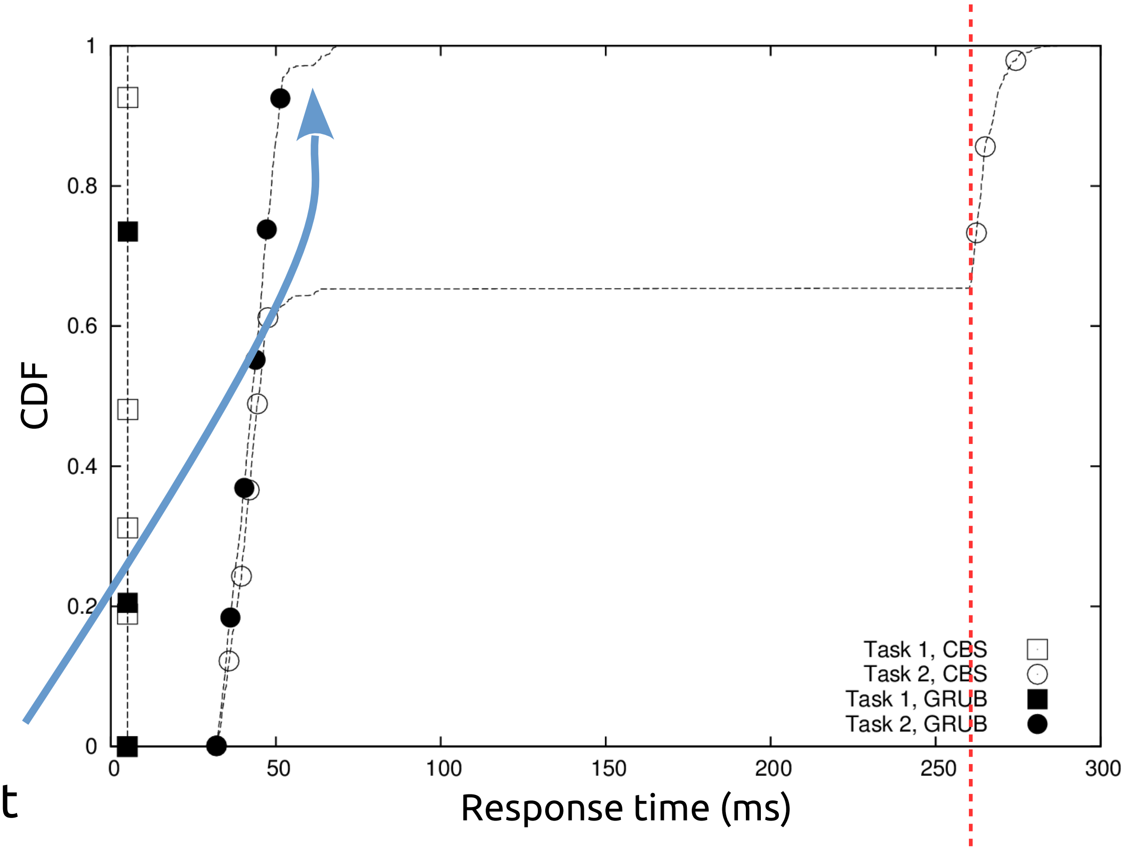
- Task1 (6ms, 20ms)

  constant execution time (5ms)

- Task2 (45ms, 260ms)

  experiences occasional variances (35-52ms)

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



T2 reservation period

* Luca Abeni, Juri Lelli, Claudio Scordino, Luigi Palopoli, Greedy CPU reclaiming for SCHED_DEADLINE, RTLWS14
http://disi.unitn.it/~abeni/reclaiming/rtlws14-grub.pdf

Clock frequency selection hints

under discussion*

- scheduler driven CPU clock frequency selection
  - schedfreq/schedutil solutions
  - each scheduling class has to provide hints
- admitted bandwidth tracking
  - worst case utilization
  - “ghost” utilization
- bandwidth reclaiming introduces per CPU active utilization tracking
  - better indication tasks' actual requirements
  - instead of donating we can decide to clock down, saving energy

* https://lkml.org/lkml/2016/3/17/420
https://lkml.org/lkml/2016/2/22/1037

Agenda

Presentation outline

- Deadline scheduling (AKA SCHED_DEADLINE)

  What is it?

  Status update

- Under discussion

  Bandwidth reclaiming

  Clock frequency selection hints

- Future work

  Group scheduling

  Dynamic feedback mechanism

  Enhanced priority inheritance

  Energy awareness

Group scheduling
future work

- one to one association between tasks and reservations
- sometime is better/easier to group a set of tasks under the same umbrella
  - virtual machine threads
  - rendering pipeline
- implement cgroups support (like for NORMAL/FIFO)
  - theory needs thinking: how can we guarantee isolation between local and global scheduler ?
  - once done it might replace FIFO/RR throttling
  - might be a practical solution for forking question

Dynamic feedback mechanism

future work

- choosing reservation parameters can be difficult (tradeoff)
  - a runtime too small ends up affecting QoS
  - a runtime too big ends up wasting CPU resource
- runtime feedback mechanism to adapt reservations to varying workloads
  - bigger time scales than bandwidth reclaiming
  - needs collaboration between kernel and userspace
  - middleware or runtime (e.g., Android) is probably best placed

Enhanced priority inheritance
future work

- move from deadline inheritance to …
- bandwidth inheritance
- similar to proxy scheduling
- boosted task runs into the donor's reservation
- not extremely easy on multiprocessors :-/

Energy awareness

future work

- in the context of energy aware scheduling (EAS*)
- meet QoS requirements in the most energy efficient way
- several things needs changing
  - introduce capacity and power awareness
  - start using energy model
  - make balancing decisions energy aware
- better integration of scheduling decisions across scheduling policies is probably required

* https://lkml.org/lkml/2015/7/7/754

Conclusions

Kernel space has already quite some features and more is in the pipeline, but…

we need more userspace adoption to foster further development (or at least more people telling us they are using it :-))

Conclusions

Kernel space has already quite some features and more is in the pipeline, but…

we need more userspace adoption to foster further development (or at least more people telling us they are using it :-))

Conclusions

Kernel space has already quite some features and more is in the pipeline, but…

we need more userspace adoption to foster further development (or at least more people telling us they are using it :-))

Conclusions

Kernel space has already quite some features and more is in the pipeline, but…

we need more userspace adoption to foster further development (or at least more people telling us they are using it :-))

Thank You!

Juri Lelli
juri.lelli@arm.com