CPU Reclaiming for SCHED_DEADLINE

Luca Abeni
luca.abeni@santannapisa.it

April 3, 2017

CPU Reclaiming: Why?

- `SCHED_DEADLINE`: allow task to execute for `runtime` every `period`
  - And if the task needs more execution time?
  - It is delayed!
  - But maybe there is some usable CPU idle time...
- Reclaiming: allow the task to execute for more than `runtime`
  - Whithout breaking guarantees for other deadline tasks!
  - Whithout starving non-deadline tasks!
    - Maximum fraction of CPU time usable by deadline tasks...

Again, Why???

- How is this related to power management???
  - If we scale the CPU frequency...
  - ...We increase the tasks' execution times
  - Need to increase the runtimes too!!!
- Knowing how much runtime we can reclaim helps to perform more informed frequency scaling
- See presentation by Juri and Claudio

CPU Reclaiming: How?

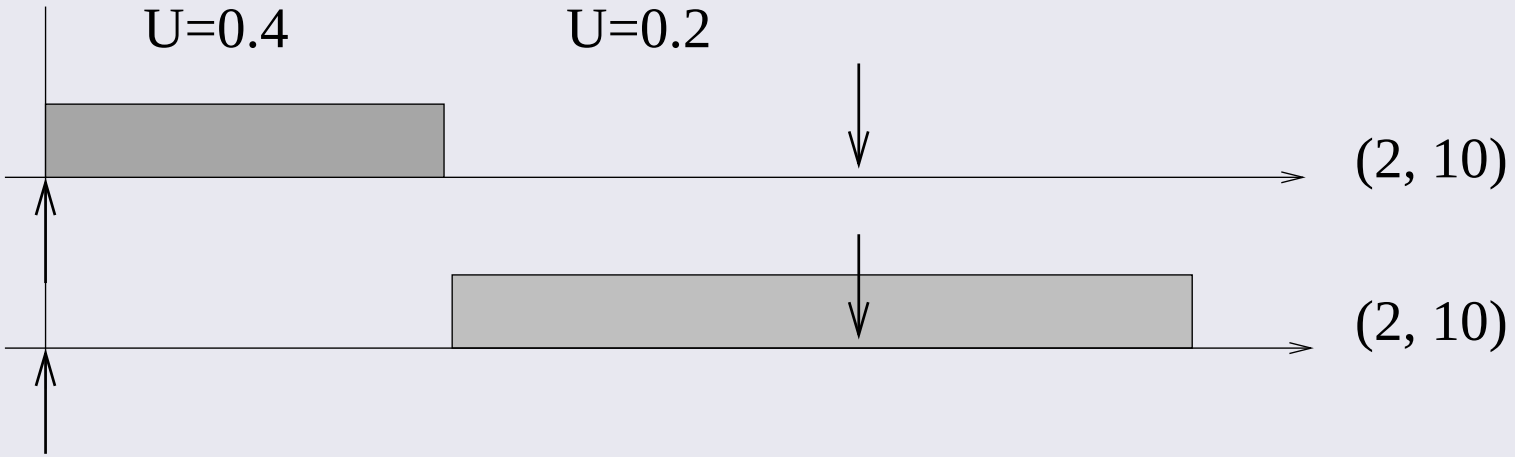
- Keep track of the fraction of CPU time of inactive deadline tasks (U^{inact})...
  - And allow to reclaim it!
- Track U^{inact} "globally" or per-runqueue?
  - In this patchset, per runqueue...
  - But a "global" alternative is also available
  - Only few patches change
- Original patchset based on U^{active}...
  - But it had fairness issues!
  - Testcase: 4 CPUs, 11 tasks with utilization 0.33
  - Cannot be easily partitioned on the 4 CPUs
  - We end up with 3 CPUs running 3 tasks and a CPU running 2 tasks

CPU Reclaiming: Utilization Tracking

- Let's see how to update U^{active} / U^{inact}...
  - Cannot be immediately updated when a task blocks
- deadline task becomes *inactive* $\rightarrow$ increase U^{inact}
  - For the runqueue where it was executing
- When does a task become *inactive*?
  - At the so called "0-*lag time*"
  - Computed when the task blocks
  - If "0-*lag time*" $\leq t$, immediately inactive
  - Otherwise, setup an "inactive timer"

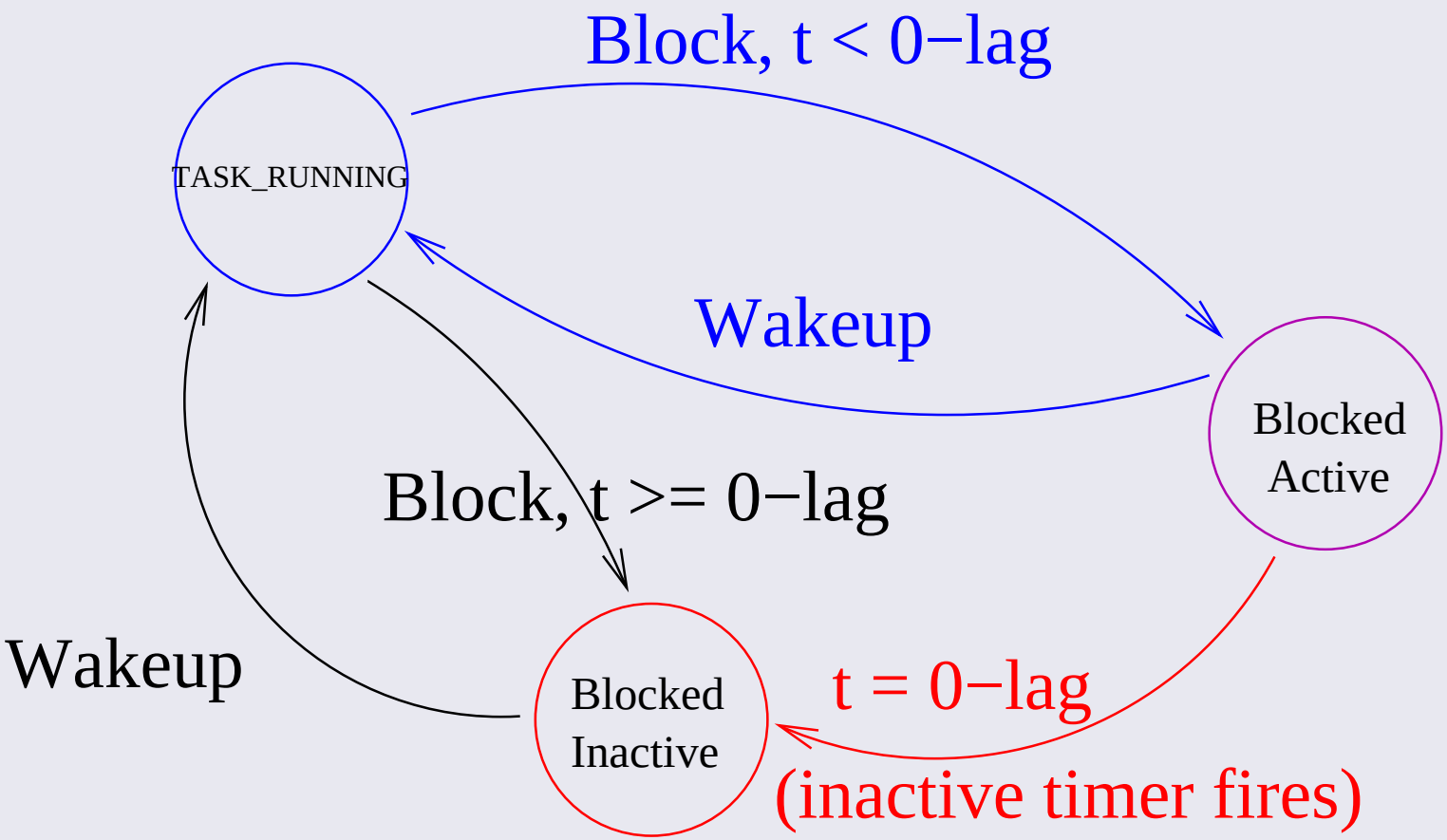
Why 0-lag?

- If $U^{act/inact}$ was updated immediately when a task blocks...
- ...We could have problems with tasks consuming their "future runtime"
  - Example: 2 tasks with runtime = $2ms$, period = $10ms$
  - First task blocking after $4.9ms$



$$\delta q = -U^{act}\delta t$$

Active / Inactive Tasks



Implementation Issues - 1

- When a task wakes up U^{inact} should be decreased
  - If the "0-lag time" already passed!
- Possible race between wakeup and "inactive timer"
  - Solved by adding a new flag (set if the "inactive timer" is pending) in the dl entity
- Wakeup on a different CPU/runqueue: might need to lock the previous runqueue

Implementation Issues - 2

- How to reclaim CPU time not allocated to deadline tasks?
  - Per-runqueue “extra bandwidth” that can be reclaimed
  - Updated when a task moves to/from deadline
- Need to iterate on all the active runqueues in the root domain!
- Interaction with CPU online / offline?
  - Would it be a good idea to initialise and clear this field in `rq_online/offline_dl()`?

Implementation Details

- Instead of tracking U^{inact}, I track U^{act} and "U_i"
  - U_i: utilization on rq i: (utilization of the deadline tasks running on rq i and of deadline tasks that blocked when running on rq i
- Then, $U^{inact} = U_i - U^{act}$
- Why this?
  - Because U^{act} or U_i ca be needed for frequency scaling
  - To minimize changes respect to previous versions of the patchset
  - This can be changed if needed

The Patchset - 1

- Patch 1: Introduce tracking of active utilization U^{active}
  - Simple (but not fully correct) tracking: increase when task enter runqueue, decrease when task exits runqueue
- Patch 2: Improve tracking of active utilization U^{active}
  - Introduce the concept of 0-lag time, the “inactive timer”, etc...
- Patch 3: Fix an old issue with the utilization of deadline tasks
  - Not related with reclaiming, but now that we have the inactive timer...
- Patch 4: GRUB accounting

The Patchset - 2

- Patch 5: Modify GRUB to reclaim only a fraction of the CPU time
- Patch 6: Enable reclaiming only for tasks that ask for it
  - Introduce a new flag in the scheduling attributes structure
- Patch 7: Introduce the tracking of RQ utilization U_i
  - Needed to compute $U^{inact} = U_i - U^{active}$
- Patch 8: base GRUB on U^{inact}
  - Fixes the fairness issue
- Patch 9: allow to reclaim more CPU time

And... What if we Want "Global" U^{active}?

- A patchset exists
- Not updated with the latest bugfixes (yet)
- Changes in patches 1, 4 and 5