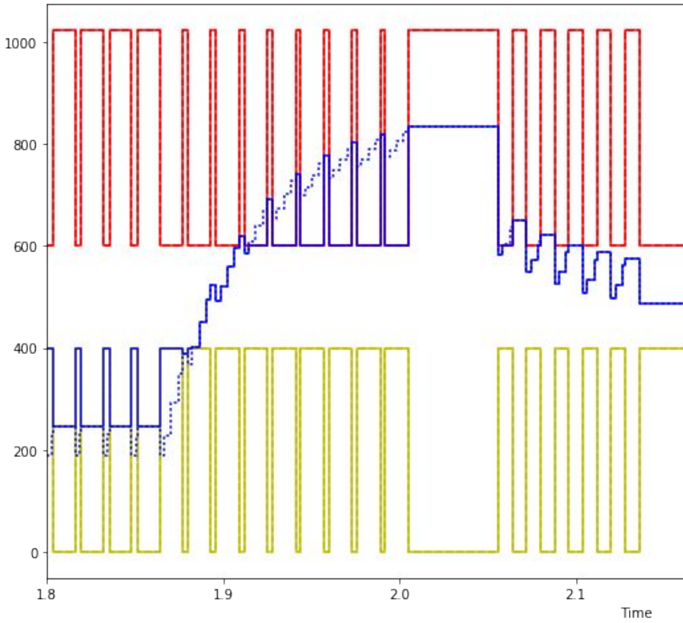
UtilClamp

Status update on Utilization Clamping support

for FAIR and RT tasks



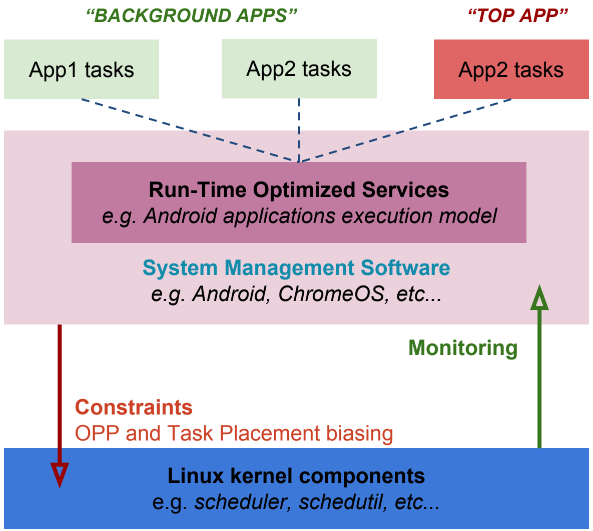
Patrick Bellasi

<patrick.bellasi@arm.com>

Introduction

What is the problem on hand?

*Feed **context aware** information about **tasks requirements***
*from **System Management Software (SMS)** to kernel-space*
*to **improve existing policies** for **OPPs selections** and **TASKs placement***



"BACKGROUND APPS"
"TOP APP"
App1 tasks
App2 tasks
App2 tasks
Run-Time Optimized Services
e.g. Android applications execution model
System Management Software
e.g. Android, ChromeOS, etc...
Monitoring
Constraints
OPP and Task Placement biasing
Linux kernel components
e.g. scheduler, schedutil, etc...

The **utilization** is already used in many decisions

- by schedutil to drive **OPP selection**
- by the (EA)Scheduler for **task placement**

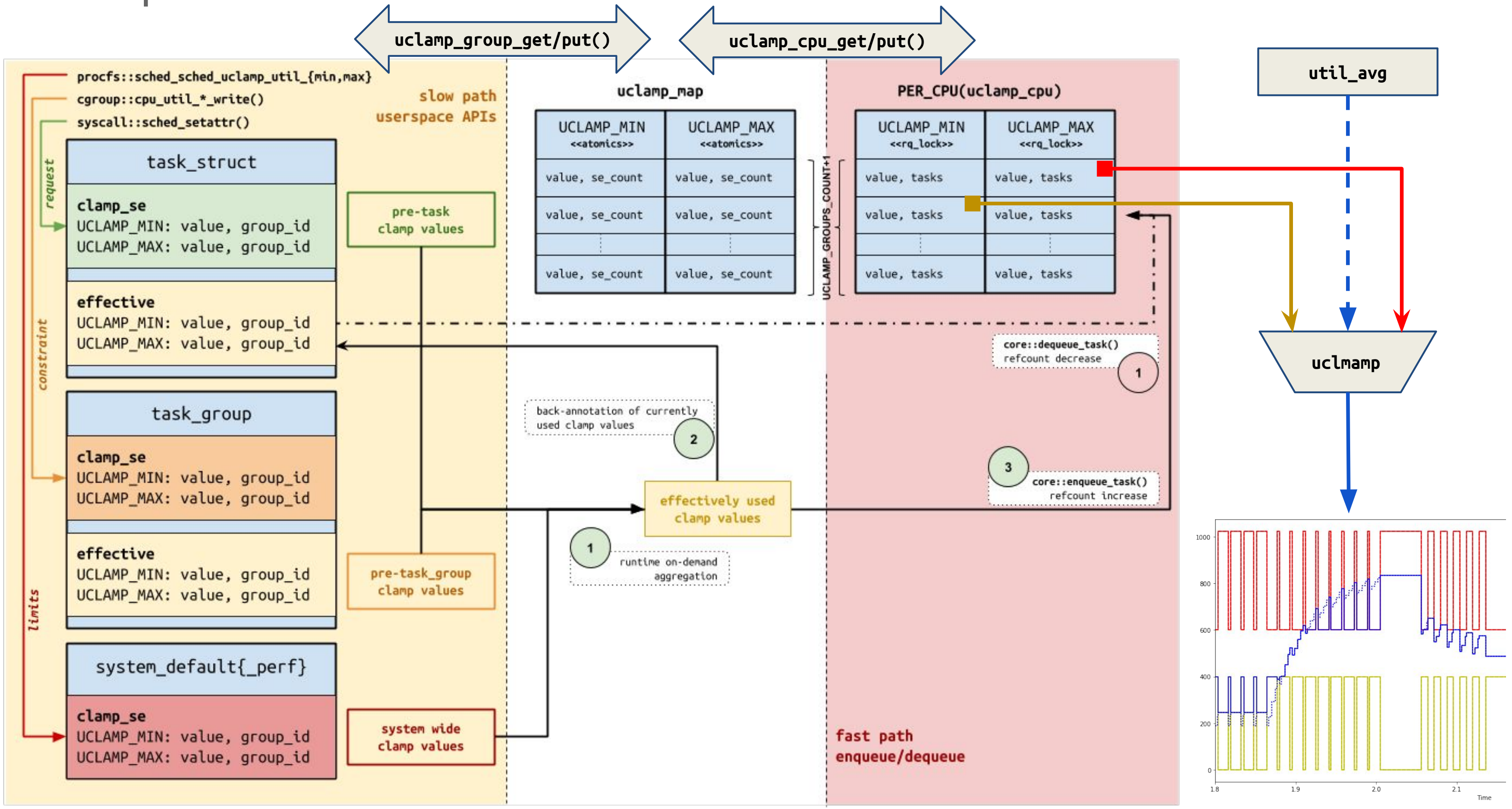
We are looking for a **per-task{group}** API

- **clamp** the utilization of each task
- **aggregate** the clamped utilization of RUNNABLE tasks on each CPU

https://goo.gl/a1i7VH [1]

Proposal

UtilClamp v5[1] in a Nutshell



uclamp_group_get/put()

uclamp_cpu_get/put()

procfs::sched_sched_uclamp_util_{min,max}

cgroup::cpu_util_*_write()

syscall::sched_setattr()

slow path
userspace APIs

task_struct

clamp_se
UCLAMP_MIN: value, group_id
UCLAMP_MAX: value, group_id

effective
UCLAMP_MIN: value, group_id
UCLAMP_MAX: value, group_id

task_group

clamp_se
UCLAMP_MIN: value, group_id
UCLAMP_MAX: value, group_id

effective
UCLAMP_MIN: value, group_id
UCLAMP_MAX: value, group_id

system_default{_perf}

clamp_se
UCLAMP_MIN: value, group_id
UCLAMP_MAX: value, group_id

pre-task clamp values

pre-task_group clamp values

system wide clamp values

uclamp_map

UCLAMP_MIN <<atomics>>	UCLAMP_MAX <<atomics>>
value, se_count	value, se_count
value, se_count	value, se_count
⋮	⋮
value, se_count	value, se_count

UCLAMP_GROUPS_COUNT+1

PER_CPU(uclamp_cpu)

UCLAMP_MIN <<rq_lock>>	UCLAMP_MAX <<rq_lock>>
value, tasks	value, tasks
value, tasks	value, tasks
⋮	⋮
value, tasks	value, tasks

1 core::dequeue_task() refcount decrease

2 back-annotation of currently used clamp values

3 core::enqueue_task() refcount increase

effectively used clamp values

1 runtime on-demand aggregation

fast path
enqueue/dequeue

util_avg

uclmamp

Main Discussion Points (1/3)

Are we heading in the right direction?

Is **bucketization** acceptable?

- user-space requests always mapped into a finite number of clamp groups

  configured at compile time, e.g. 10-20, as a linear sub-division of the max capacity
- from use-cases on hand we do not expect many different boost/clamp values

  clamp groups mapping ensure to use only the minimum number of clamp groups actually required

Are **system defaults** acceptable?

- system_default clamps for FAIR tasks, restrict task-specific and task group clamps

  exposed as (root only writable) /proc/sys/kernel/sched_uclamp_util_{min,max}

  by default: util_min=0 and util_max=SCHED_CAPACITY_SCALE
- system_default_perf clamps for RT task

  by default: util_min=util_max=SCHED_CAPACITY_SCALE

Is clamping **acceptable for RT** tasks?

- entirely optional framework, no overheads on !CONFIG_UCLAMP_TASK
- even when compiled in, system_default_perf defaults to always running at max freq

  still allows to improve energy efficiency for certain RT tasks on mobile systems

Main Discussion Points (2/3)

How far are we?

Is the **effective** aggregation acceptable?

- scheduler: compute the actual clamp value at enqueue time

  a caching mechanism is possible if we should consider that an overhead
- cgroups: transparently track the most restrictive clamp between a group and its parent

  subgroups can always change their clamps

  hierarchical updates ensure to always propagate and use the max value

What's the best **merging** strategy?

- keep refining core bits and merge those before cgroup integration...

  risk of data structures not suitable for a smooth integration in the cpu controller
- ... or update the full patchset until both core bits and cgroup support are ACKed?

  safer solution but will required more time

Main Discussion Points (3/3)

What are possible future extensions?

Add a **timer-based release** semantic ?

- event-based clamp set, timeout-based clamp reset

  touchboost is an example use-case already used in Android
- it can potentially be used to implement features like the **iowait boost**

  with the advantage of being a the per-task / user-space defined hint

Add a **generic kernel-space API** to access clamp groups ?

- drivers and/or firmware can be interested in asserting clamp values
- we can take advantage of a unified and well defined interface to aggregate user/kernel-space clamps

  kernel-space clamps can provide a restriction to user-space clamps

  which aggregation policy makes sense will be defined by a single "framework"
- kind-of similar to **pm_qos** but more cpu and task specific and limited to clamp values

  maybe it could make sense to just add util clamp metrics to pm_qos?

Thanks for the discussion

That's all... for Today