uclamp in CFS: Fairness, latency, and energy efficiency

Linux Plumbers Conference 2023

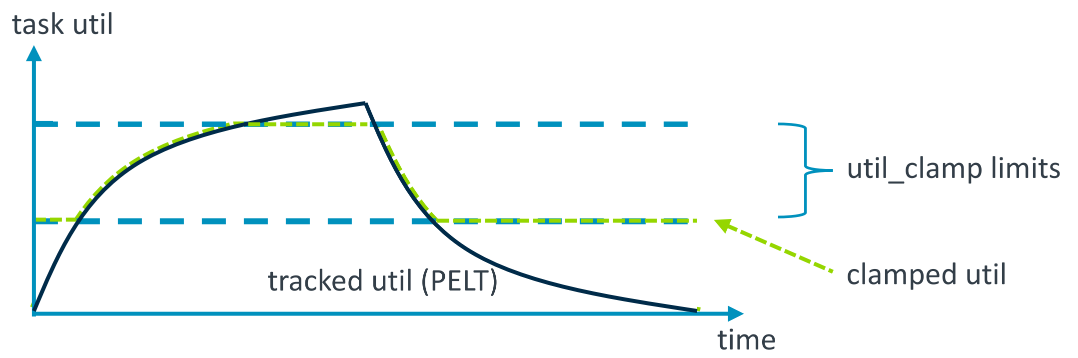
Power Management and Thermal micro-conference

Morten Rasmussen, Dietmar Eggemann, Hongyan Xia

14. November 2023

Motivation

- CFS task placement and schedutil DVFS policy is based compute demand derived from CPU utilization tracked on per-task basis (PELT).
- Utilization clamping, uclamp_{max, min}, offers a user-space interface to bias and override a task's compute demand for scheduling and DVFS decisions.



- Main issues with uclamp:
  - Aggregation of per-task uclamp settings for each CPU runqueue, i.e. sum() vs. max().
  - Implementation complexity: Add another PELT-based signal at 'source' and propagate, or compute when needed.

uclamp aggregation: max()

- Upstream uclamp currently considers uclamp settings as performance hints describing desired throughput rate when executing, **not** actual throughput (rate * cpu_time).
  - uclamp aggregates per-task uclamp settings using max() aggregation.

Runqueue

util_avg: 100
min: 250
max: 1024

util_avg: 100
min: 250
max: 1024

cfs_rq util:	100+100	= 200
cfs_rq util_min:	max(250, 250)	= 250
cfs_rq util_max:	max(1024, 1024)	= 1024
cfs_rq clamped util:	**clamp(200, 250, 1024)**	**= 250**

- Capacity is shared with any other tasks on the same CPU runqueue.
  - This is particularly unfortunate for uclamp_max.

Runqueue

util_avg: 1024/5
min: 0
max: 512

5 always-running tasks

cfs_rq util:	5*1024/5	= 1024
cfs_rq util_min:	max(0, 0, 0, 0, 0)	= 0
cfs_rq util_max:	max(512, 512, 512, 512, 512)	= 512
cfs_rq clamped util:	**clamp(1024, 0, 512)**	**= 512**
Capacity per task:	**512/5**	**= 102**

uclamp aggregation: max() issues

- Current implementation has per-task uclamp settings applied to rq utilization.
- Advantage:
  - No additional PELT-derived signals to maintain, clamp applied when needed.
- Disadvantages:
  - Max-clamped task's utilization may not represent true compute demand at all:
    - For tasks running alone, utilization is likely to over-estimate demand.
    - For co-scheduled tasks, tasks' utilization may under-estimate demand.
  - Difficult to distinguish UCLAMP_MAX throttled CPU and CPU running at its peak.
    - https://lore.kernel.org/all/20230916232955.2099394-2-qyousef@layalina.io/
  - Max-clamping impact on rq utilization causes problems when tasks with different max-clamps are queued together.
    - Causes frequency spikes.
  - Tracking max clamp setting for all tasks on rq doesn't scale well. Currently implemented using buckets.
  - Difficult to reason about throughput of max-clamped tasks.
- Cause of current issues:
  - uclamp not applied at 'source' and virtually impossible to reconstruct at rq level.
  - Max aggregation doesn't provide a clear policy for balancing clamped tasks.
- Possible solutions:
  - Max-aggregation filter + minimum capacity-per-task (unclear).
  - uclamp sum() aggregation with clamping applied at source creating a new PELT-derived signal.

uclamp users: Android

- Given the current gaps in the mainline uclamp implementation it is unclear if uclamp is widely used.
  - Android (on Pixel 8) uses mainline uclamp implementation but for tasks only.
- Google essentially implemented uclamp sum() aggregation at task group level.
  - Android features CONFIG_USE_GROUP_THROTTLE and CONFIG_USE_VENDOR_GROUP_UTIL.
  - Implemented using Android Vendor hooks.
  - We actively try to raise interest in Google to get engaged into the mainline discussion of uclamp sum() aggregation.

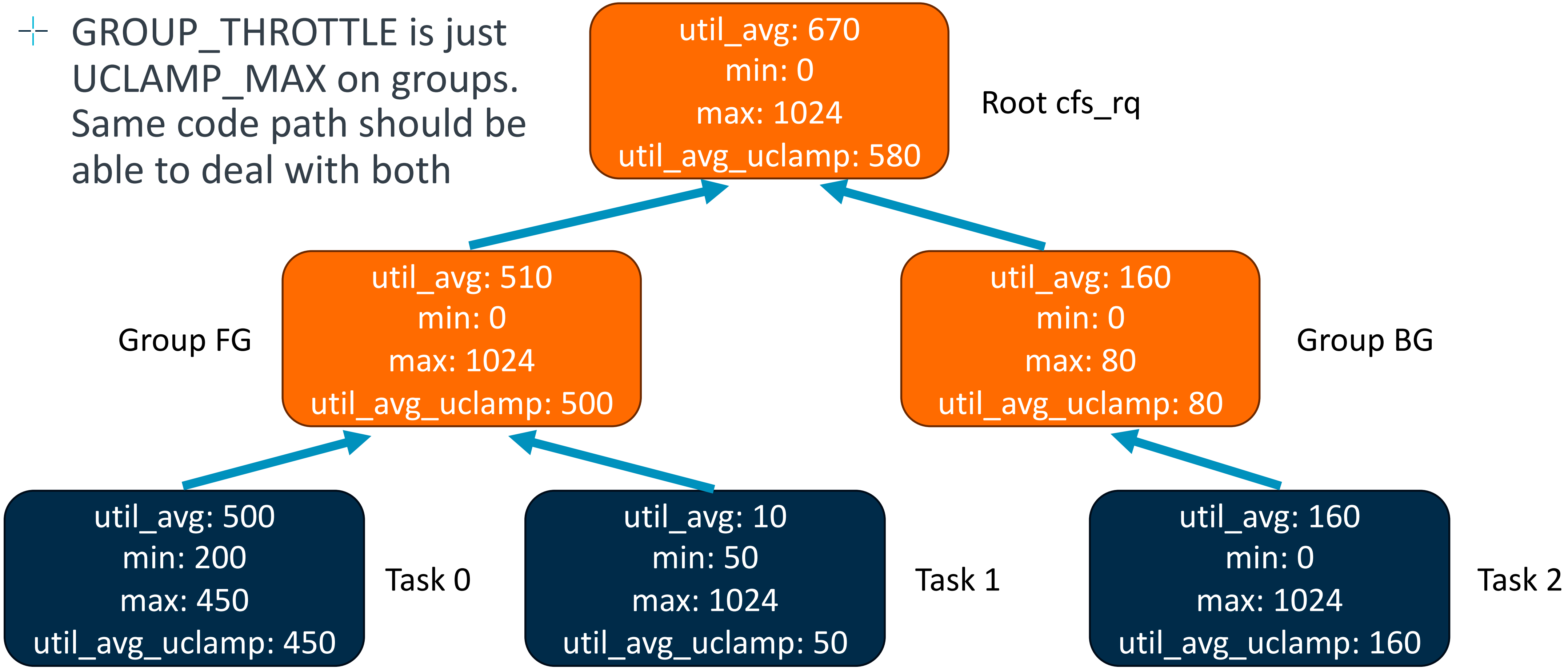
- Can we agree on a useable upstream uclamp implementation?

uclamp: sum() aggregation RFC

- RFC: Learn from Android changes and consider sum() aggregation:
  - RFC on LKML: https://lore.kernel.org/all/cover.1696345700.git.Hongyan.Xia2@arm.com/
  - No changes to user-space API and fundamental goals remain the same.
  - Add new PELT-derived signal: util_avg_clamped on tasks and propagated to root rq.
  - Significant code complexity reduction: +341/-751 LOC.
  - Pre-liminary results look good, see patch set cover letter.

Overview of uclamp sum aggregation

- GROUP_THROTTLE is just UCLAMP_MAX on groups. Same code path should be able to deal with both

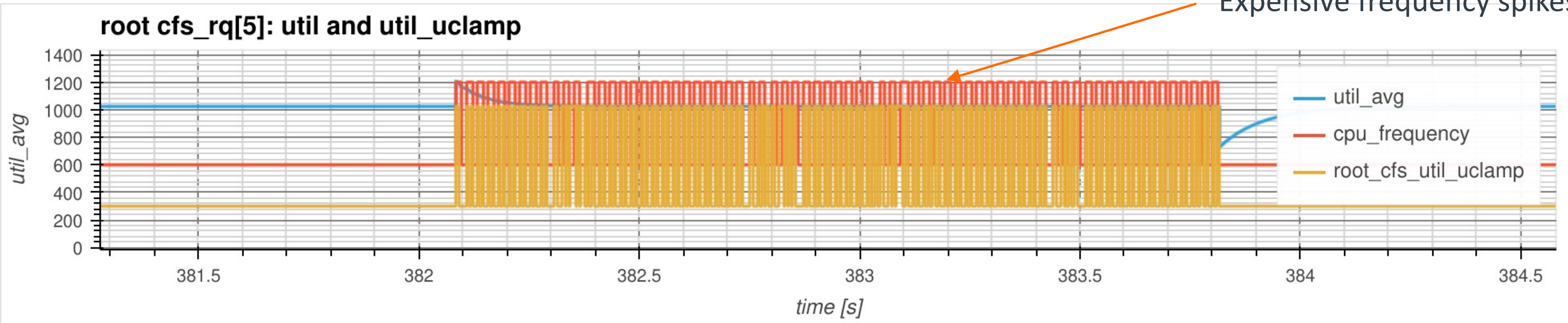


Comparison (max vs. sum): Frequency Spikes

- Scenario:
  - Always-running task with UCLAMP_MAX of 300 (30%).
  - Joined by a task with 40% duty cycle and default UCLAMP_MAX (1024) (100%)

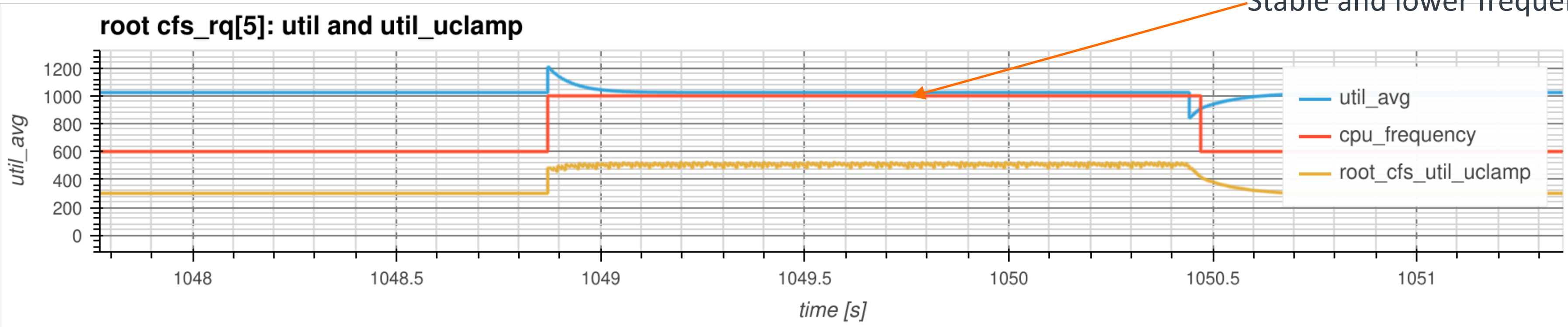
Expensive frequency spikes

max()



Stable and lower frequency

sum()

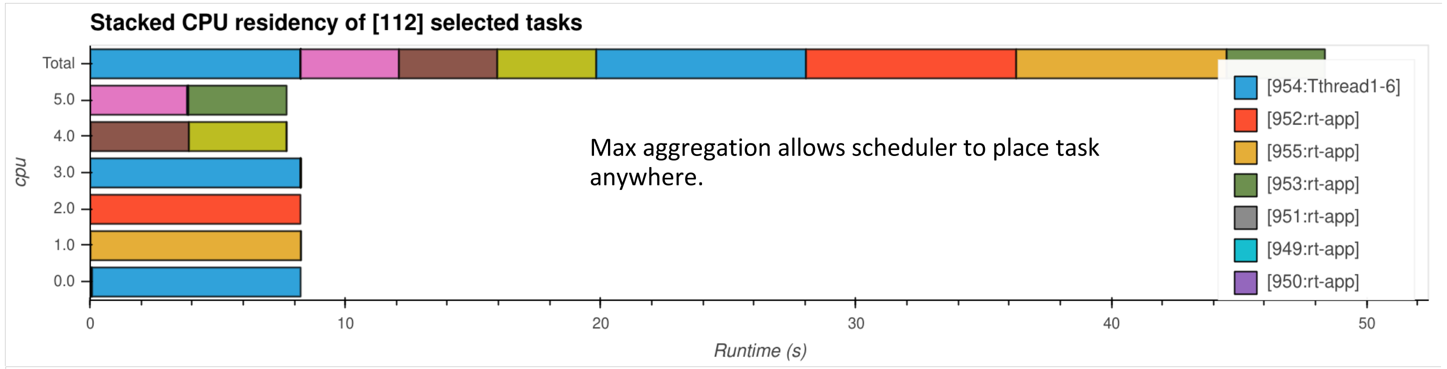


arm

Comparison (max vs. sum): Task placement

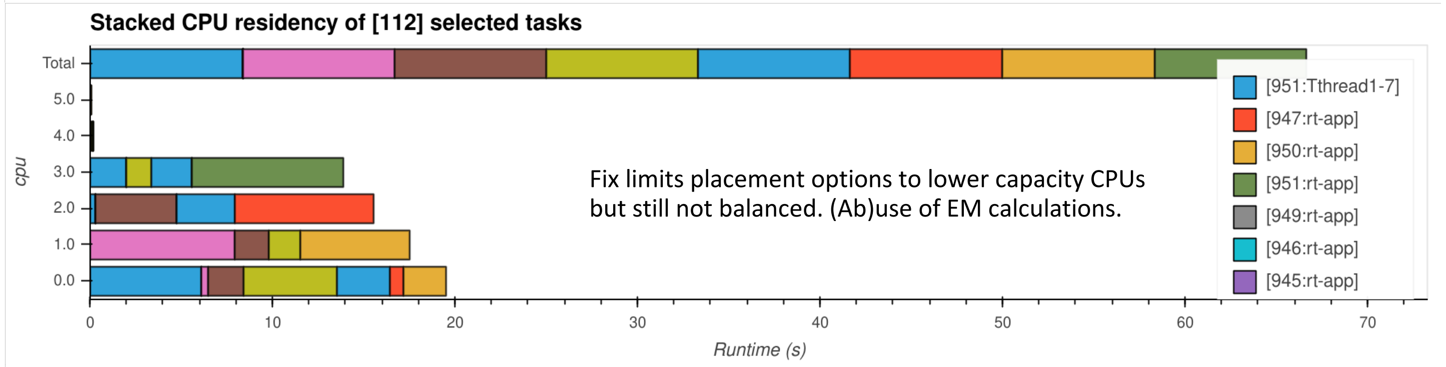
- Scenario:
  - 8 tasks uclamp_max = 120

Upstream: max()

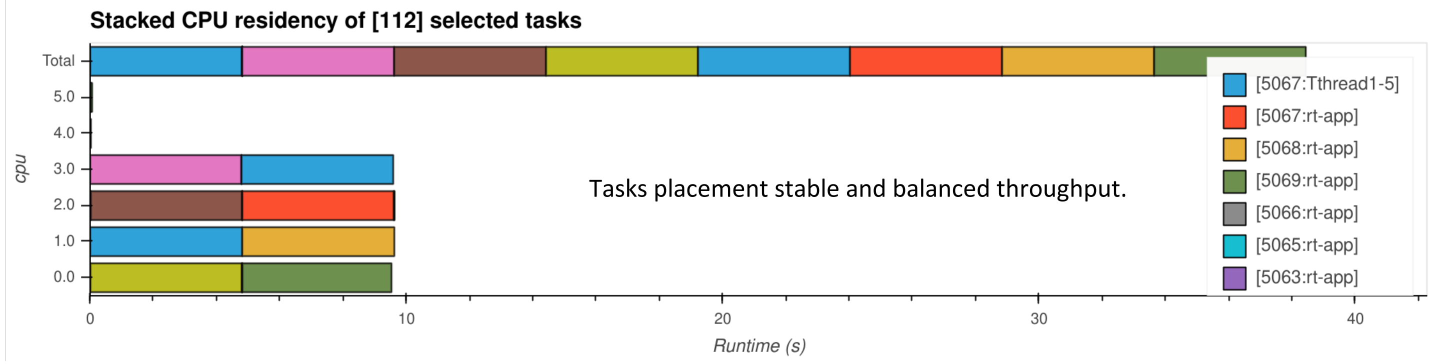


Upstream: max() + fix

Patch set: 'Set max_spare_cap_cpu even if max_spare_cap is 0'



RFC: sum()



RFC results: Simpler code and good initial results

```
include/linux/sched.h            |  13 +--
init/Kconfig                     |  32 --------
kernel/sched/core.c              | 316 ++++++------------------------------------------------------------
kernel/sched/cpufreq_schedutil.c |  19 ++---
kernel/sched/fair.c              | 354 ++++++++++++++++++++++-----------------------------------------------------
kernel/sched/pelt.c              | 146 +++++++++++++++++++++++++++++++--
kernel/sched/rt.c                |   4 -
kernel/sched/sched.h             | 208 +++++++++++++++--------------------------------------
8 files changed, 341 insertions(+), 751 deletions(-)
```

- Better uclamp with less than half of the code
- Example: Jankbench 75.44% jank reduction and 0.9% energy increase, sum vs. max aggregation
- Example in `util_fits_cpu()`:
  - From more than 100 lines (including tons of comments) to just one line:

```
return fits_capacity(util, capacity);
```


Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה