

# EAS: SchedTune

## Working Towards the Next Generation

Patrick Bellasi  
<patrick.bellasi@arm.com>

# Agenda

- Original Goals and Motivations
- SchedTune v2 current status
- Alternative proposal: SchedTune v3
- Future Steps

# SchedTune: Design Goals

Provide a simple, central tunable for

*energy saving vs performance boosting*

Bias **OPP selection** and **tasks placement**

- provide schedfreq-schedutil with behaviours similar to other governors  
*e.g. interactive, performance*
- support EAS to trade-off **energy saving** for **performance boosting**

Fosters the collection of sensible information from **informed run-times**

- to support better task scheduling decisions
- by providing a simple yet effective API to middleware like Android

# SchedTune: Current Status (i.e. what's in use)

RFC v2 posted on LKML <sup>[1]</sup>

- supporting only OPP boosting but based on schedutil integration

Full solution available in ACK v3.18 <sup>[2]</sup>

- supporting task biasing via EAS integration
  - in find\_best\_target() for !is\_big\_little targets*
- small refinements to support either PELT or WALT utilization signal
- using additional attribute to better support latency sensitive tasks

Further fixes and improvements in MSM v3.18 <sup>[3]</sup>

- available in partner's msm-google kernel tree
- improved performance index definition

[1] <https://lkml.org/lkml/2016/10/27/503>

[2] <https://android.googlesource.com/kernel/common/+android-3.18>

[3] <https://android.googlesource.com/kernel/msm/+android-msm-marlin-3.18-nougat-mr1-eas-experimental>

# SchedTune: Improved Performance Index

Performance index discounting for **potential delay sources**

$$\text{Perf\_idx} = \text{SpeedUp\_idx} - \text{Delay\_idx}$$

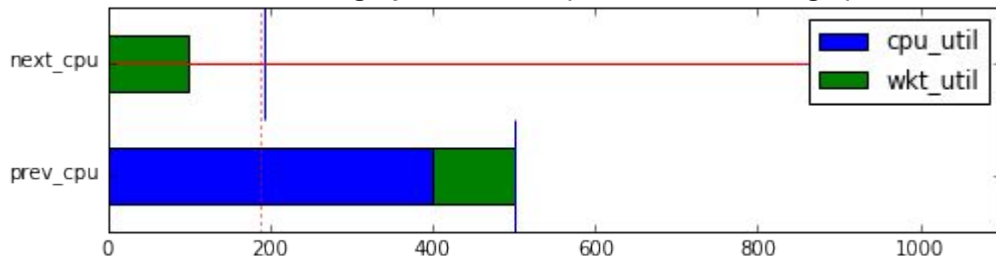
- estimate of “how fast” the task will run

$$\text{SpeedUp\_idx} = \text{cpu\_boosted\_capacity} - \text{task\_util}$$

- discount all the latency treats (e.g. co-scheduling, Hi-Prio tasks, blocked-load, IRQ pressure, etc.)

$$\text{Delay\_idx} = 1024 * (\text{cpu\_util} - \text{task\_util}) / \text{cpu\_util} \text{ [1]}$$

10% tasks waking up, 10% boost (~90 utilization margin)



Next\_cpu preferred depending on:

- prev\_cpu utilization and blocked load
- boosted CPU's capacity

# SchedTune: Android “Specific” Optimizations

Some extensions have been evaluated during the Pixel’s tuning exercise

- addition of **prefer\_idle** for latency sensitive tasks
- experiments using **negative boosting**

**Energy-efficiency** and **Low Latencies** are both required for different class of tasks

- depending on task status, e.g. TA vs BG

	<b>Neg Boosting</b>	<b>No Boost</b>	<b>Boosting</b>
<b>Energy Efficiency</b>	BG	BG / SYS_BG	FG (non TA)
<b>Lower Latency</b>	Camera	FG	TA

# SchedTune: Main Complaints from LKML/LPC <sup>[1]</sup>

## Introduction of a **new CGroup controller**

- the boost value is **affecting** the availability of **CPU's bandwidth**
- Tejun&PaulT proposed to integrate this concept into the **existing CPU controller**  
*this should support a more coherent view on what is the status of the CPU resource*

## Enforcing (by design) a "**flat hierarchy**" of boosted tasks

- a flat hierarchy does not match the expected "**generic behaviors**" for **CGroup interface**
- such a controller cannot be easily extended to **support CGroup v2** configuration

## The request for a **single knob** has been kind-of demoted

- some implementation details currently do not allow to **grant the required boost values**
- boosting support is really required only for **mid-to-big deltas**  
e.g. small tasks with big boosting, but not the big tasks with small deltas  
*a threshold based implementation could be potentially good enough*

# SchedTune: Alternate Proposal Principles

Original Concepts	Mapping within the CPU Controller
Boost value	Using the existing <b>cpu.shares</b> attribute <ul style="list-style-type: none"><li>- by default tasks have a 1024 share</li><li>- boosted tasks gets a share &gt;1024 (more CPU time to run)</li><li>- negative boosted tasks gets &lt;1024 (less CPU time to run)</li></ul>
OPP biasing	Add a new <b>cpu.min_capacity</b> attribute Tasks in the group are granted to be scheduled on a CPU which provides at least the required minimum capacity
Negative boosting	Add a new <b>cpu.max_capacity</b> attribute Tasks in the group are never scheduled on a cpu with CPU capacity higher that this value (at least while they are alone on that CPU)
CPU selection and prefer_idle	The <b>cpu.shares</b> value can be used as a “flag” to know when a task is boosted e.g. is <code>cpu.shares &gt; 1024</code> (or threshold) we look for an idle CPU The <b>cpu.min_capacity</b> can also bias the selection of a big CPU The <b>cpu.max_capacity</b> can also bias the selection of a LITTLE CPU
Latencies reduction	Tasks with higher <b>cpu.shares</b> value are entitled more CPU time and this turns out to give them better chances to get scheduled by preempting other tasks with lower shares.  <b>NOTE:</b> the CPU bandwidth not consumed by high <b>cpu.shares</b> value tasks is still available for tasks with lower shares.

## CPU utilization clamping

<https://lkml.org/lkml/2017/2/28/355>

## Task Placement

## Performance Boosting



# SchedTune v3: Implementation Details

CPU's keep track of capacity constraints

- for all RUNNABLE tasks
- using RBTrees to keep `task_struct` ordered

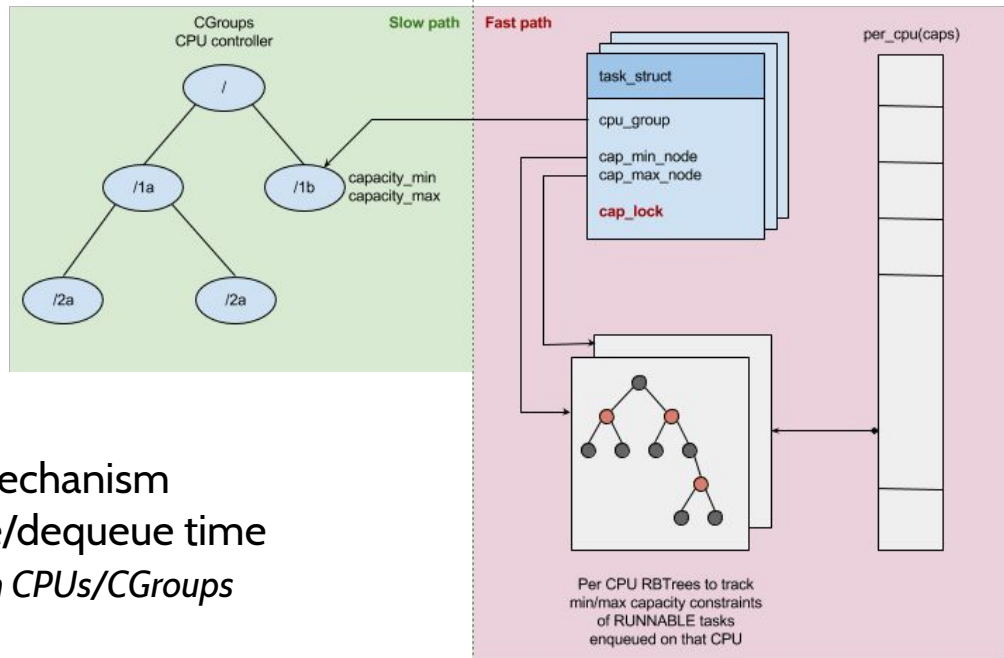
Tasks ordered based on capacity constraints enforced by their CGroups

- simple accounting and aggregation mechanism
- insertion/removal ops just at enqueue/dequeue time

*free support for tasks migrations between CPUs/CGroups*

Main features

- capacity clamping tracked by the core scheduler
- *support for both FAIR and RT tasks*
- No limitations on number of “boost groups”



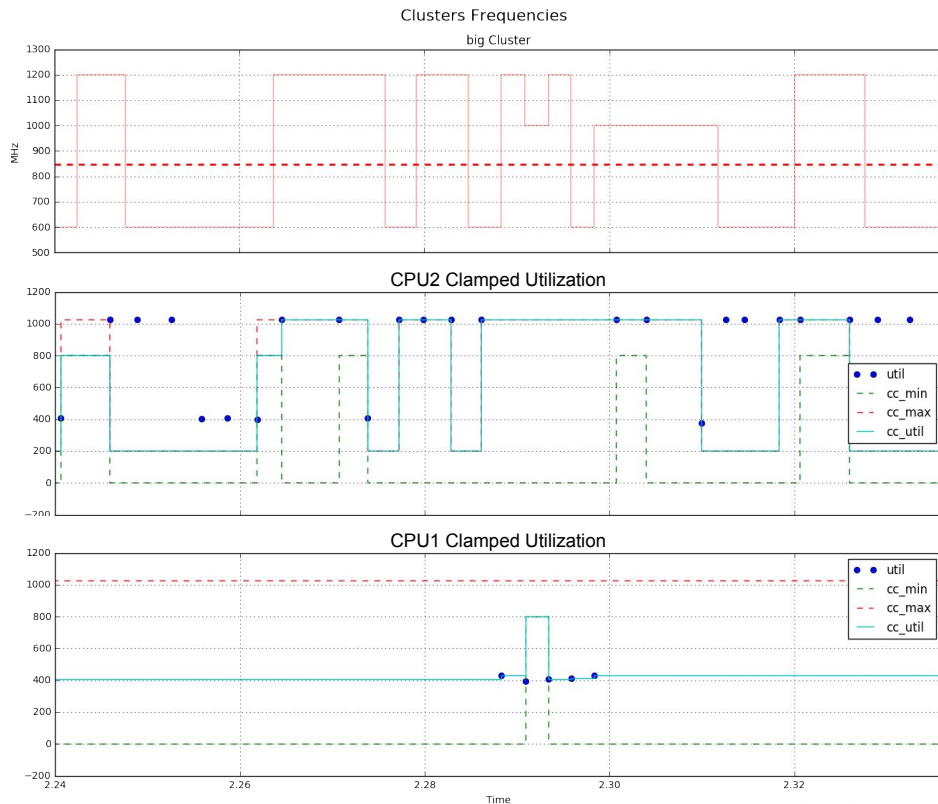
# SchedTune v3: Capacity Clamping Validation

Functional validation performed on JUNO R2 boards

- using this rt-app synthetic scenario

10x10% background tasks  
capacity\_max=20%  
cpumax=0x4

1x10% top-app task  
capacity\_min=80%  
cpumax=0x4



# SchedTune v3: Shares Benefits on Latencies

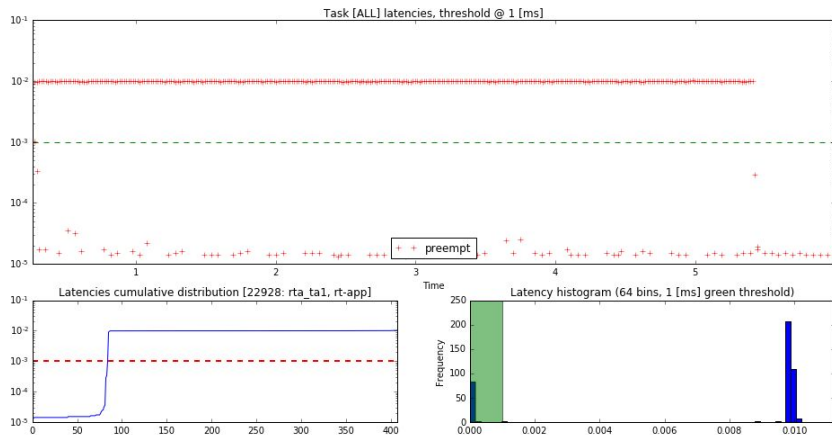


Fig.1 - Shares: BG=1024, TA=1024

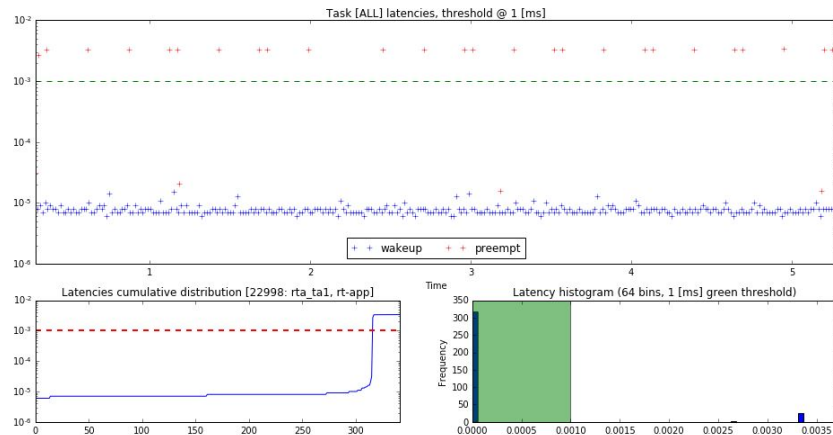


Fig.2 - Shares: BG=2, TA=1024

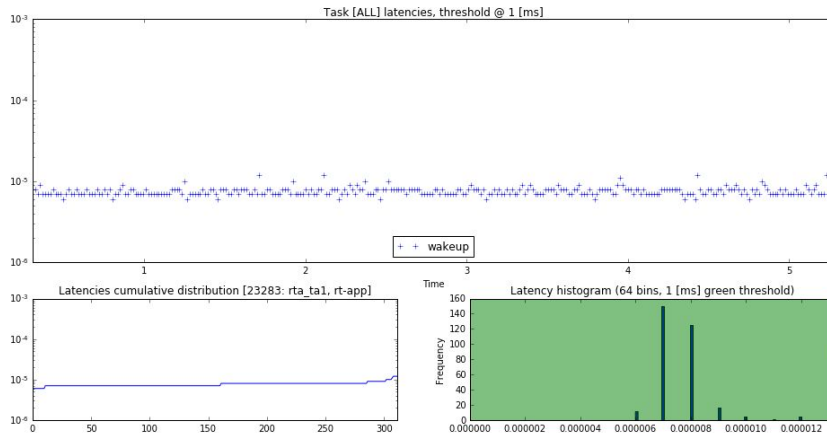


Fig.3 - Shares: BG=2, TA=10K

# SchedTune v3: Works in Progress

## Complete task placement biasing

- remap **prefer\_idle** to a suitable check condition on **cpu.shares** value
- the performance index will not be added in the first instance

Integrate v3 (possibly beside v2) in EAS r1.3 for ACK 4.4

## Complete the AOSP userspace integration

- refactor/cleanup current sched\_policy<sup>[1]</sup>
- extends full task classes to cpuctl
  - BACKGROUND
  - SYSTEM\_BACKGROUND
  - FOREGROUND
  - TOP\_APP
- update both cpuctl and cpuset at each policy setting/updating
- **android/platform/system/core**
  - rootdir/init.rc
  - libcutils/sched\_policy.c
- **android/platform/frameworks/av**
  - media/audioserver/audioserver.rc
  - media/mediaserver/mediaserver.rc
  - camera/cameraserver/cameraserver.rc
- **android/platform/frameworks/base**
  - services/core/java/com/android/server/UiThread.java
  - cmds/bootanimation/bootanim.rc
  - core/java/android/os/Process.java

# SchedTune v3: Future Advanced Topics

## Experiment with CFS bandwidth controller

- investigate the possibility to replace the usage of cpusets with a proper and more complete configuration of the CPU bandwidth controller
- should optimize parallelization of background tasks, especially when there are not foreground and/or top apps running

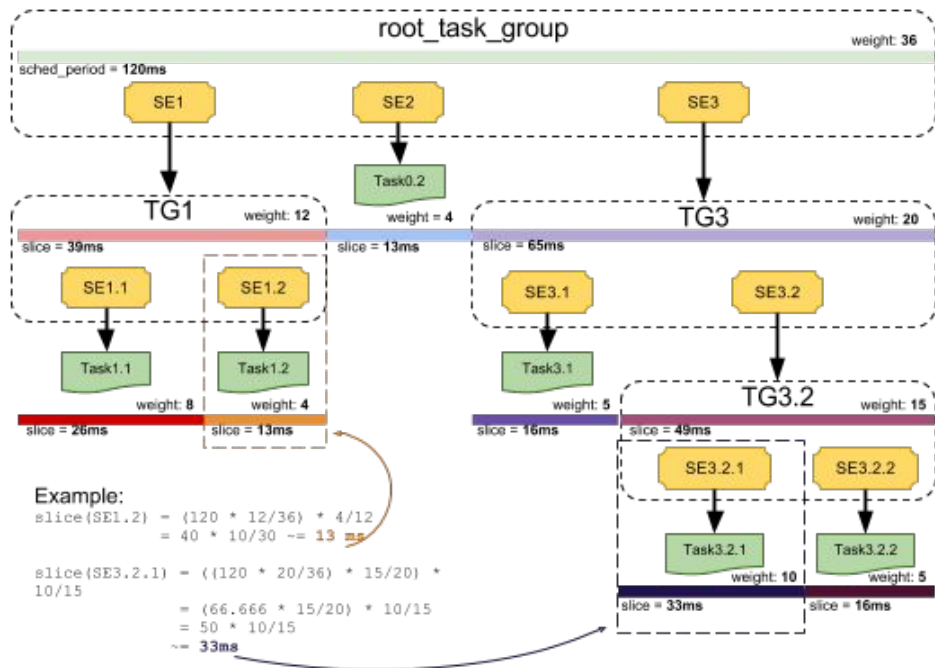
## Using per-task CGroups instead of task classes

- this is expected to reduce overheads related to moving tasks around
- better match the most “classical” usage of the CGroup interface, i.e.  
*“Organize Once and Control [1]”*

# Backup Slides

# SchedTune v3: How Shares Works?

Similarly to how SE's priority defines the “weight” of a TG, and thus its slice time



Note: The weight of a TG is the sum of weights of which SE are part of the that group

- Used to repartition the scheduling latency (SL)  
`/proc/sys/kernel/sched_latency_ns`  
10ms by default in AOSP
- A quota of SL, proportional to its share, is assigned to each SE  
never smaller than:  
`sched_min_granularity_ns`