Android EAS Patches
2019 Update

BKK19 Power Hacking Session

Chris Redpath
Android EAS Patches

Android-4.14

• 87 Patches
• 52 files changed, 7211 insertions(+), 417 deletions(-)

Android-4.19

• 75 Patches, 19 android-only
• Total: 55 files changed, 4478 insertions(+), 364 deletions(-)
• Android only: 27 files changed, 1963 insertions(+), 103 deletions(-)
Android EAS Patches – Functional Grouping

What we call EAS is made up of a few functional groups, some upstream or in-progress:

- Energy Model & Energy-cost wakeup placement plus other things - around 80% of EAS pieces already upstream*  
  - Upstream in 5.0, enablers in 5.1 for all common cpufreq drivers  
  - ~80% of the functionality, not necessarily 80% of the LoC

- Thermal Capping  
  - Upstream replacement in progress by Linaro – will replace ~110 LoC in ACK and has potential to unify a handful of different vendor-specific equivalents

- Performance Boosting (schedtune)  
  - Upstream replacement in progress from Patrick Bellasi @ Arm called util_clamp – will partly replace ~1500 LoC  
  - Util_clamp does not replicate everything schedtune provides
Android EAS Patches – Functional Grouping (2)

- There are still some bits of functionality for which there is not yet an agreed upstream alternative or where the android parts are still in development:
  - Find_best_target
  - Schedtune prefer_idle & boosted task placement
  - A Schedtune-hold equivalent for maintaining active frequency boosts
  - Load Balance tweaks
  - Tracing & debugging
  - Schedutil Governor tweaks
  - Wake_q depth-optimised wakeup placement
EAS wakeup placement strategies

Find_best_target has multiple algorithms depending upon schedtune status of tasks

- minimise latency OR minimise energy impact OR maximise available performance
- encourages ui-sensitive threads from the current application onto ‘reserved’ cpus to reduce pre-emption (working in conjunction with schedtune, cpusets and android framework)

upstream EAS algorithm:

- For each frequency domain which can provide sufficient cpu capacity, select the lowest-utilized cpu where the task is allowed to run
- Compute the energy required to run the task on each candidate cpu
- Select the lowest-cost option

Corresponds well to the fbt non-prefer-idle & unboosted cases – these concepts do not exist upstream
find_best_target deprecation plan

1. Add prefer_idle and use schedtune boosted task util signals in the mainline path for tasks with that attribute*
   - The patch which adds this also adds an alternative to the c-state tracking using exit latency, further reducing the amount of out-of-tree code

2. For android-4.19, change sched_features so that find_best_target is not used by default
   1. Both these two are currently done in android-4.19
   2. Test results are linked from this change: https://android-review.googlesource.com/c/kernel/common/+/929494/5

3. For future android versions (see experimental/eas-dev), don’t include find_best_target at all

*this is a mitigation plan until we have util_clamp upstream and come up with a prefer_idle alternative we prove works
Schedtune Functionality (prefer_idle)

Prefer_idle is a schedtune cgroup attribute which we set on top-app and foreground cgroups so it will be going away when schedtune is removed in favour of util_clamp.

3 options under consideration – main attention on option 1.

No changes expected for any current android branches.

Replacement Option 1: Remove it entirely

- We already look for max spare capacity with the simplified EM, which is one of the prefer_idle behaviours
- We will be using boosted utilisation for this placement, so can avoid little CPUs with appropriate boost levels
- In android-4.14, going for idle cpus ignores blocked load so this might actually help us
- One of the main tasks, surfaceflinger, has become an rt task in O (new)
Schedtune Functionality (prefer_idle)

Backup options if we fail to satisfy ourselves we no longer need it:

Backup option 1: Replace with task policy & priority
  • Combine these in some way to infer a binary ‘prefer_idle’ attribute, then behave similarly to now

Backup option 2: Push for something similar upstream based on util clamp
  • This is the fallback to the fallback, not considered much yet
Schedtune Functionality (boosting)

• Util clamp min/max util levels will replicate boosting but in a different way
  • Schedtune is proportional, util clamp is absolute

• Tuning will have to be done from scratch

• Upstream util clamp will only influence frequency selection
  • Android has been using schedtune boosted utilisation for cpu placement in wakeup as well
  • Might be able to upstream if we prove benefit in Android
Schedtune Functionality (schedtune hold)

• Holding is a hack in schedtune which keeps cpu rq boosting active for a short time once tasks go away
  • Intended to cover gaps where we have RT tasks which wake up/sleep often enough that the frequency change rate limits would prevent changing to match task requirements

It’s quite simple to implement if you have a list of active boost/clamps

• Only need an expiry timestamp updated when a task is activated
• Check the expiry timestamp whenever we ask for the current cpu boost/clamp
• Remove the boost/clamp if the current time is past the timestamp
• No timers required, only one patch
• Feels quite hacky – but covers cases where tuning the system to ‘fix it properly’ isn’t possible
• Could potentially be integrated into schedutil instead – may be more acceptable upstream
Load Balance Tweaks

- There are 5 of these
  - ANDROID: sched/fair: Also do misfit in overloaded groups
  - ANDROID: sched/fair: Don't balance misfits if it would overload local group
  - ANDROID: sched/fair: Attempt to improve throughput for asym cap systems
  - ANDROID: sched: Enable idle balance to pull single task towards cpu with higher capacity
  - ANDROID: sched: Prevent unnecessary active balance of single task in sched group

Most of these cover problematic placement we saw in synthetic tests – some of which no longer exist as of 5.1-rc2

General plan is to evaluate each one, push for an upstream solution if it can still happen, and in either case drop the patch from the next android

We expect to be able to eliminate all of these from next android kernel
Tracing & Debug

• A significant amount of the android-only code in android-4.19 is related to debug and testing
• Upstream does not want to add additional trace_events as they form a new userspace ABI
• Ideas?
  • We’re looking into eBPF, but struggling to replicate the event tracing easily
  • Would rather not just maintain an out-of-tree patch set, but that’s possible

Note: We can run our LISA tests without many of these, but actually understanding use cases is hard without additional tracing
Schedutil Governor Tweaks

Right now we have a couple of changes in android:

- Use cpu_rq signals for RT task utilisation rather than go to max
- Split up/down rate limiting to make it easier to increase than decrease

One more actively being worked upon in Arm:

- Use the EM to identify higher frequencies we could use without paying too much energy penalty

Commonly-done changes in devices:

- Something similar to the hispeed stuff from interactive
- Finer granularity tuning

Any ideas/feedback/hypothesis?
Wake_q optimised wakeup placement

Currently in android-4.14 & 4.19:

- When the wakeup primitive in kernel knows how many waiters are about to be completed.
- Pass that knowledge through to the wakeup balancer in the scheduler
- Use that to override any energy saving and go for a wider spread of task placement

Useful for benchmarks and potentially other things upstream, but impact was fairly small in testing and it needs the exact correct set of config flags & usecase

Plan for now is to drop this in next android, and work on upstreaming it

- Not currently offering any benefit in android as system configuration usually prevents it being available
- Quite controversial since it can’t cover all use cases and not always clear what the benefit is
Thank You
Danke
Merci
谢谢
ありがとうございます
Gracias
Kiitos
감사합니다
धन्यवाद
תודה