QUALCOMM®

PELT vs Window tracking and EAS on SMP multi-cluster

Agenda

PELT vs window tracking

- Introduction: PELT
- Introduction: Window tracking
- Task load tracking comparisons
- CPU load tracking comparisons
- Real world use case

EAS on SMP multi-cluster

- Energy model
- Spreading tasks
- Real world use cases results

PELT vs Window based load tracking (WinLT)

PELT refresher

(excerpts from LKML)

- Load tracking on a per-entity basis –
  - Load sum for a cfs-rq is the sum of its children's load averages.
  - Previous solution tracked load average per cfs-rq

- Load is separated into runnable and blocked load averages.
  - Blocked and runnable load is decayed in the same way.

- Replaces windows with fine grained tracking

- Current PELT math decays load such that load from 32ms ago contributes around 50% towards current execution.

se->avg.load_avg_contrib
Load average contribution of a single task.

cfs_rq->utilization_load_avg/utilization_blocked_avg
Actual CPU utilization of running tasks and recently blocked tasks – aggregated per cpu runqueue.

Window based load tracking (WinLT) refresher

- Keeps track of N execution windows per task
- Based on the N samples available per-task, a per-task "demand" attribute is calculated which represents the CPU demand of that task. This calculation is policy defined and the current policy is max (average, most-recent)
- Windows of observation for task activity are synchronized across CPUs.
- Scheduler guided frequency provides aggregate load from all tasks that ran in the most recently finished window.
- About ~2500 lines of code on top of the 3.18 scheduler

Window based load tracking (WinLT) refresher

Separate load tracking statistics for CPU vs Tasks

- Policy differentiation between frequency management and placement.
- Account wait-time for task demand, but not for CPU load
- Mobile workloads on Big.Little/SMP can gain an advantage from this separation
  - For example, starting a new task – WinLT task demand can be set high/low enough for placement, without actually influencing the frequency of the target cluster.

task->ravg.demand: per-task "demand" attribute

prev_runnable_sum: aggregate demand on a CPU from all tasks which executed during the most recent completed window. This is done using the same windows that the task demand is tracked with.

	PELT	WinLT
Load Tracking	Load is accounted using a geometric series that effectively halves load contribution from 32ms prior to current time	Load is accounted with a policy that observes load average during the past N windows. Policy selection allows more accurate tracking of mobile workloads
Blocked load/utilization tracking	Load is decayed as part of a runqueue statistic when the task is blocked, task load average is decayed appropriately when task is enqueued again.	Blocked time is essentially “null” time – load contribution is removed from runqueue sum/average statistics.
Blocked load restoration	Runqueue statistics include blocked load/utilization at all points of time.	Load contribution is restored to RQ statistics when the task becomes runnable again.
Effect on load reporting to frequency governor (e.g., interactive governor)	Statistics reported to or used by governor cause slow ramp-up. Requires EAS-like boost during tick and sched_tune boosting Smoother load profile reported due to slow ramp up and ever decaying load.	Faster ramp-up especially helpful to real world mobile use case. No explicit boosting may be necessary for most usecases. More bursty load reporting. May require careful window stats policy selection.

PELT and WinLT

Unit Test

- Workload: Single thread running on single core that executes an integer/fp CPU bound workload for 100ms and sleeps for 80ms.

- Trace points: Observe at every tick, enqueue, dequeue and pick_next_task_fair:
  - PELT - load_avg_contrib/util_avg and
  - WinLT - rvg.demand/prev_runnable_sum

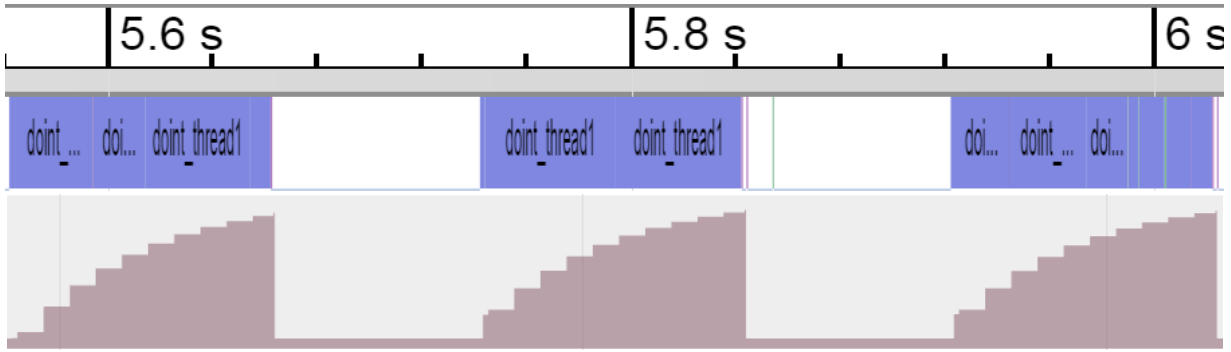
- Use a very small task that executes every 20ms to keep statistics updated and observable.

- Tracepoint sets statistic to zero when the task is dequeued

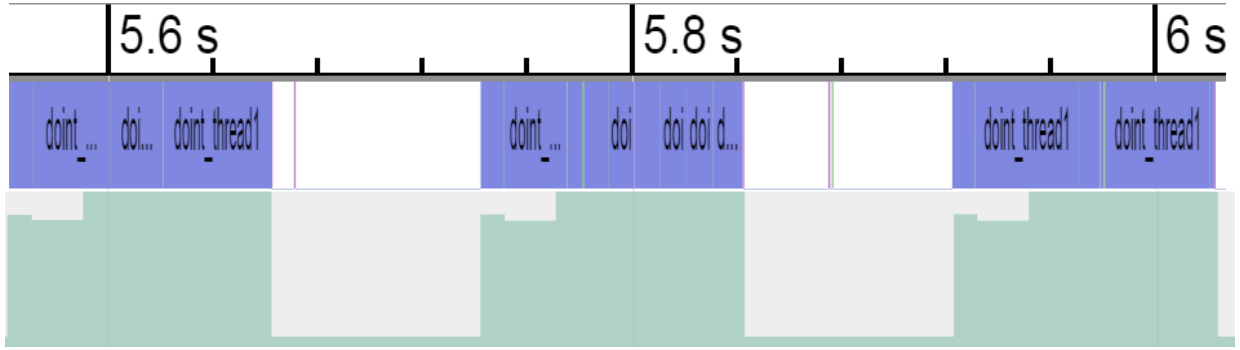
PELT and WinLT

Task Load Tracking Ramp Up

PELT: takes more than 100ms to report full load (1023). Every time the process sleeps, history is lost and the process has to re-execute to “regain” its previous load value (load_avg_contrib)

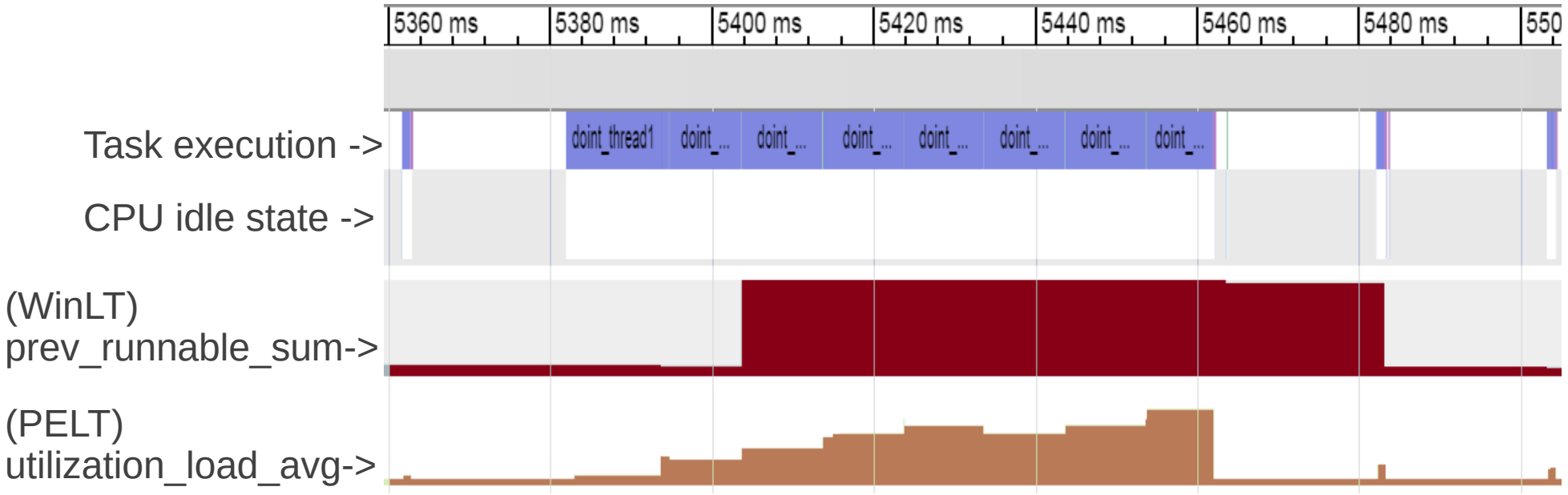


WinLT: With a window size of 10ms and no loss of history, load reported reaches max (10ms) within 3 windows every time the process executes. (ravg.demand)



PELT and WinLT

CPU Load Tracking Ramp Up

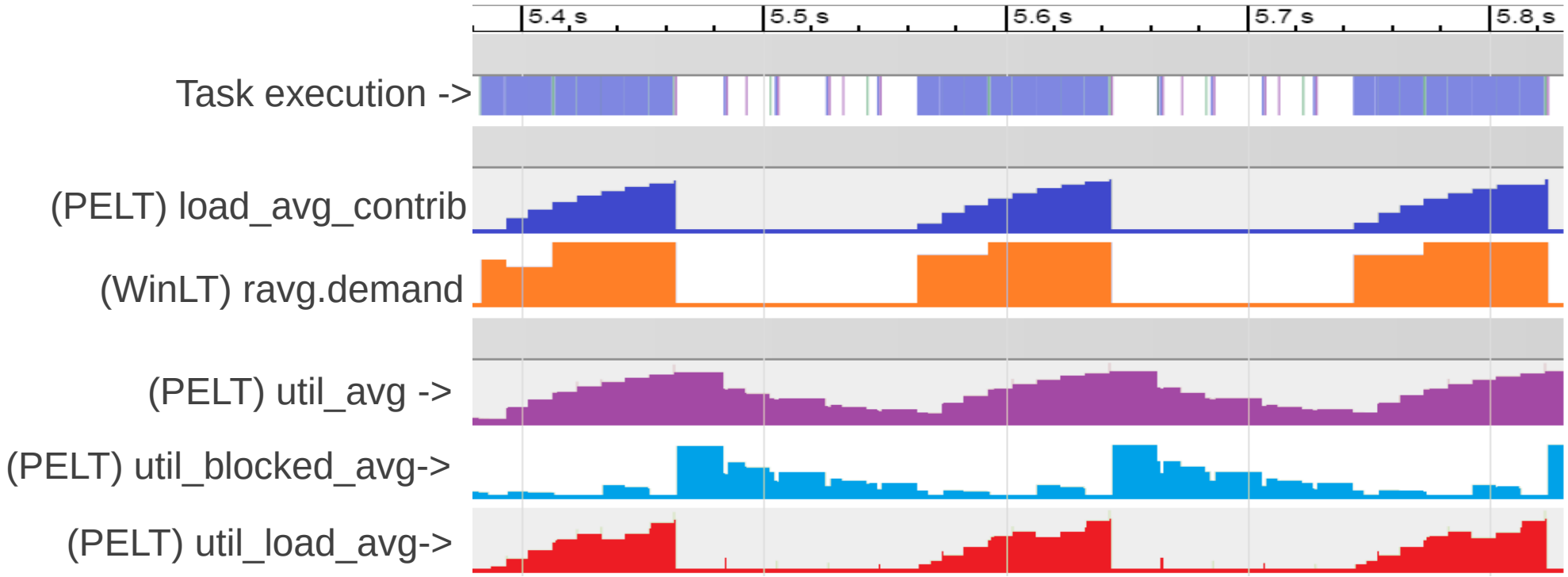


Observations:

1) WinLT tracking ramps up much faster than PELT, which gradually rises but doesn't report max load in this usecase.
2) It might seem like PELT is dropping the utilization much faster once the task sleeps, but it is actually just transferred over to utilization_blocked_avg....

PELT and WinLT

Task Load Tracking Ramp Down

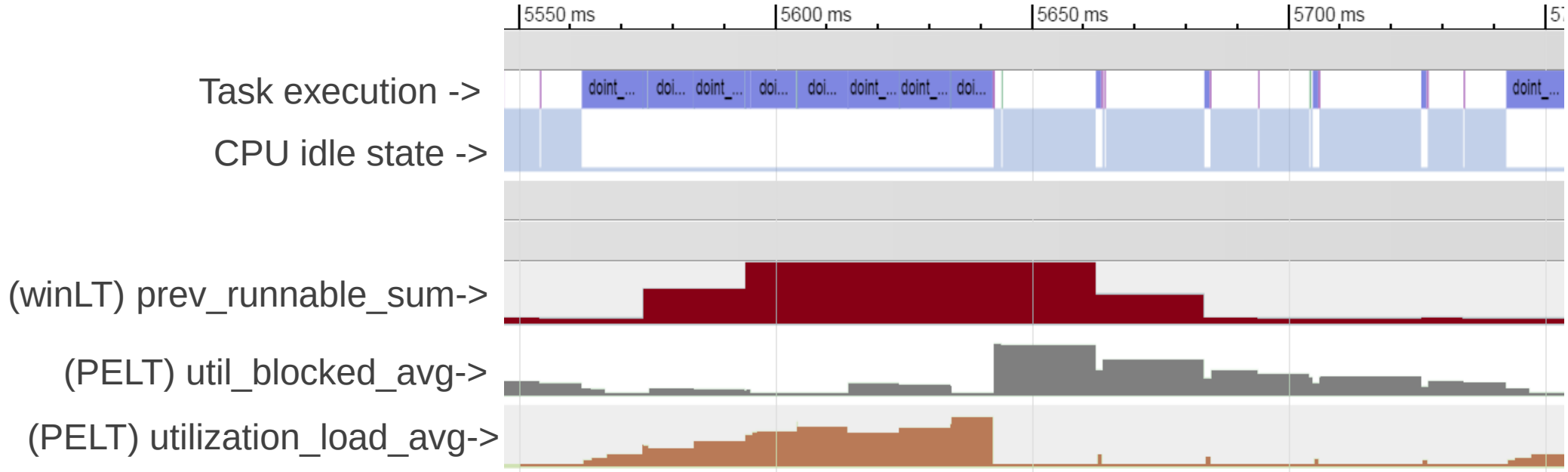


PELT:
Once the workload sleeps, the load average is moved over to blocked load and decayed, still contributing to util_avg. It takes several ms longer for the load to decay.

WinLT :
Once the workload sleeps, the load is removed from the runnable sum and average statistics, and updated at the end of the window.

PELT and WinLT

CPU Load Tracking Ramp Down



Observations:

1) WinLT tracking ramps down much faster than PELT. Since PELT tracks blocked task utilization, and that is also part of the metric reported to sched-freq, the frequency ramp down time can be much longer
2) It might seem that PELT is keeping history, but that history is completely lost by the time the task wakes up again, and thus is of little use to ramping up frequency again.

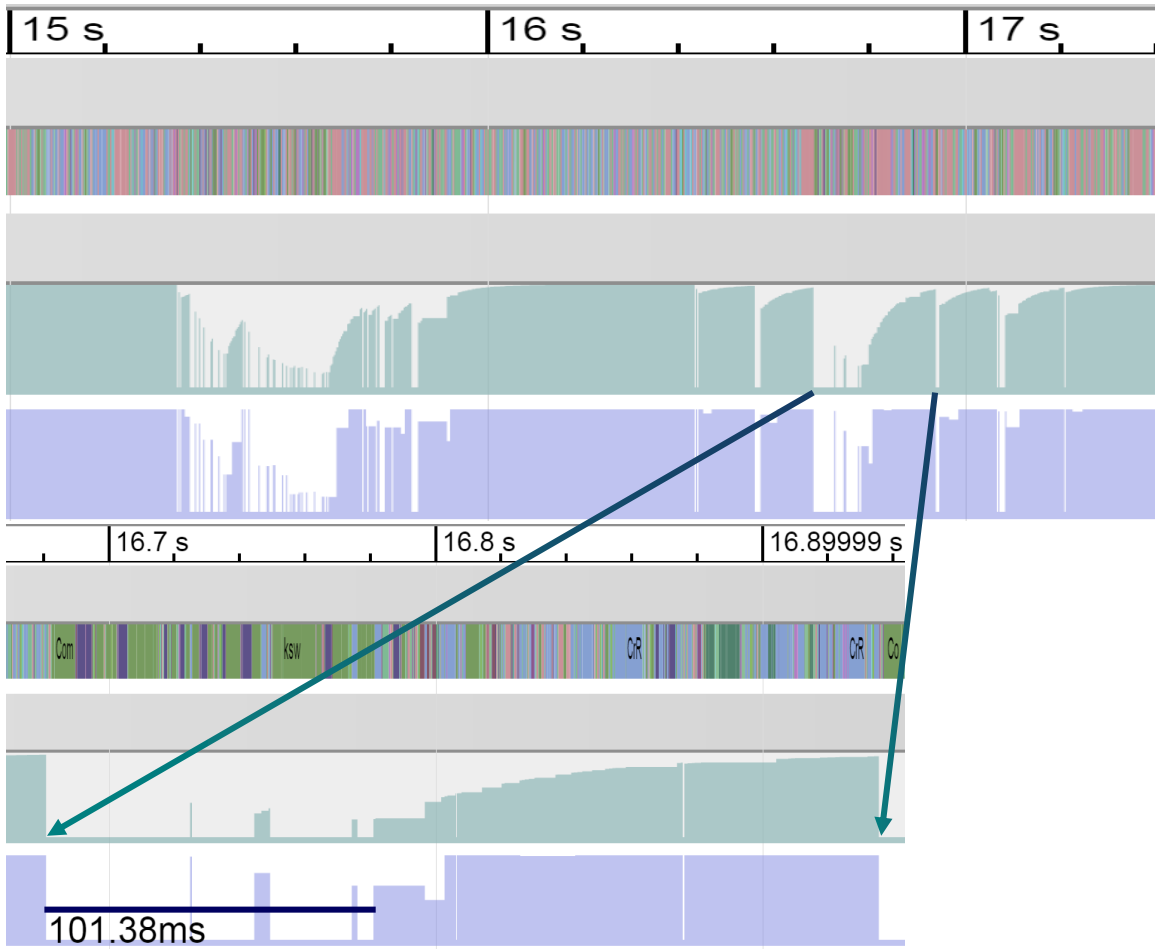
Real World Task Load Tracking

Chrome browser scroll (single core, fmax)

- Open up engadget.com and scroll
  - Track stats for CrRendererMain (one of the primary workload threads of the Chrome browser)

PELT - load_avg_contrib ->

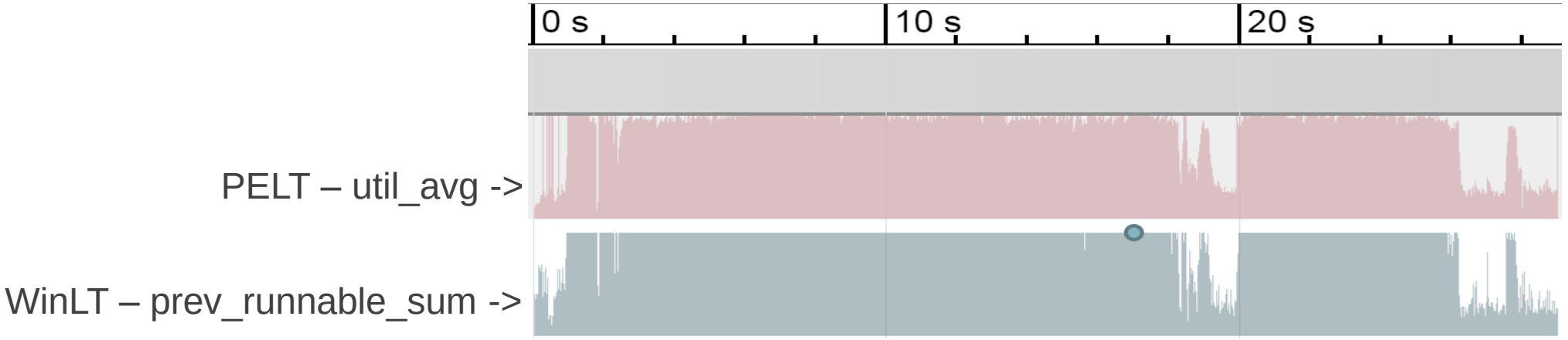
WinLT – ravg.demand ->



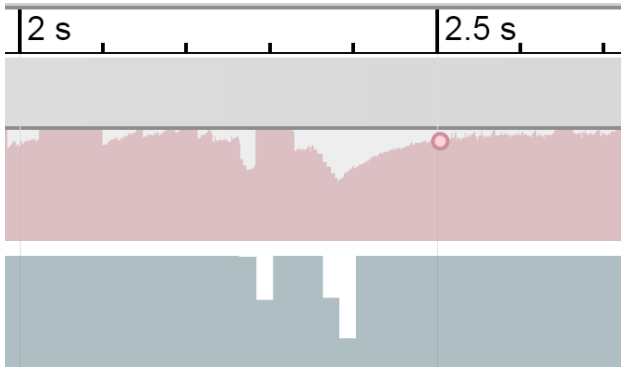
The zoomed in view on the right shows how PELT decays load to a low value after a 100ms sleep requiring the browser process to run for >120ms before reporting max load once again. WinLT ramps up to max within two windows (40ms)

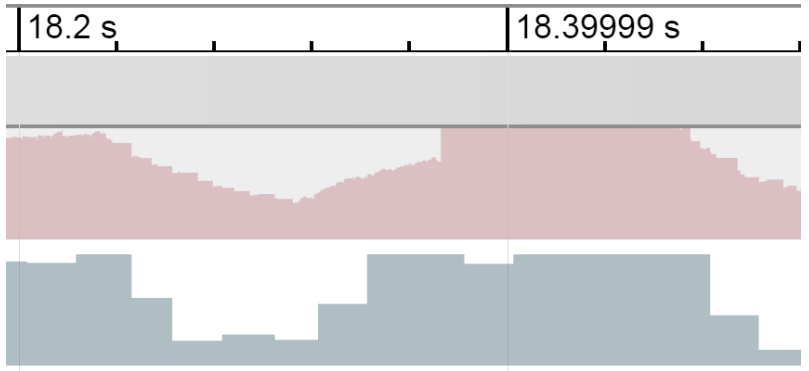
Real World CPU Load Tracking

Chrome browser scroll (single core, fmax)



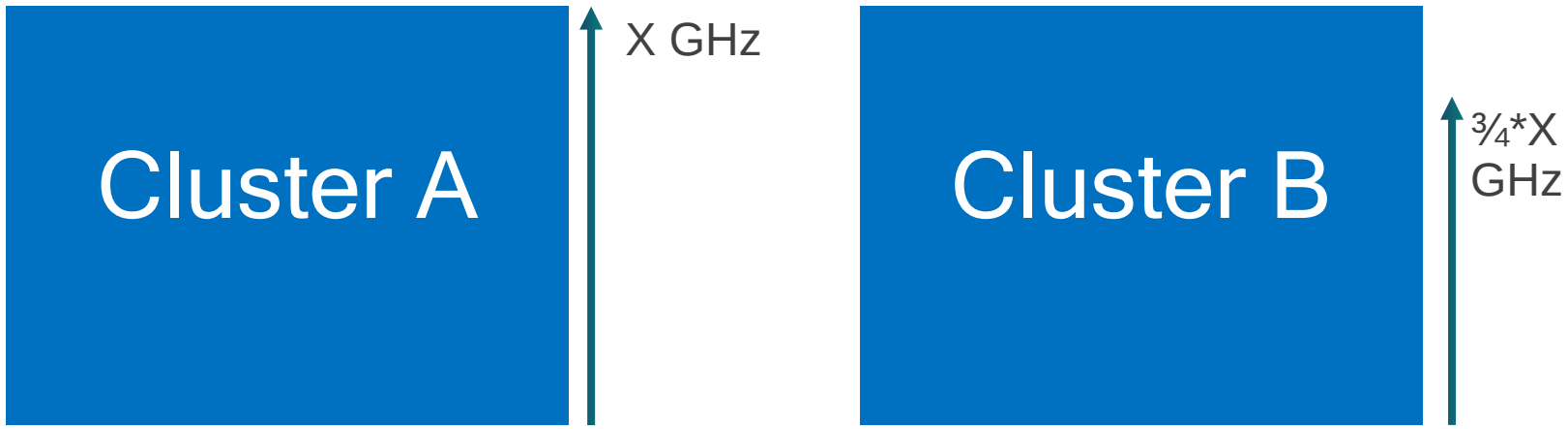
The bird's eye view above shows a similar profile! But the zoomed in view of certain parts shows that WinLT ramps up and down faster for the reasons listed in previous slides. This does result in fewer janks and better power when these statistics are reported to or used by the frequency governor.





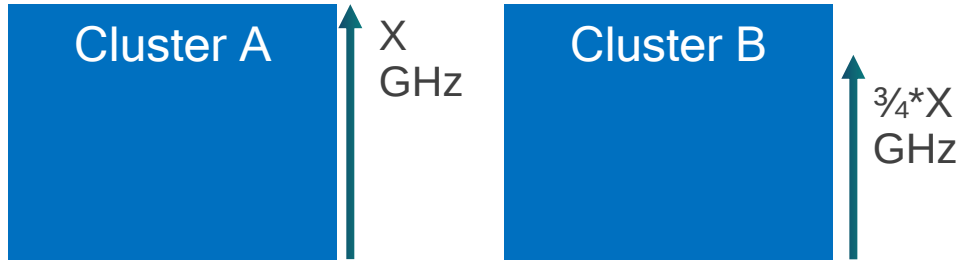
EAS on SMP multicluster

SMP multi-cluster

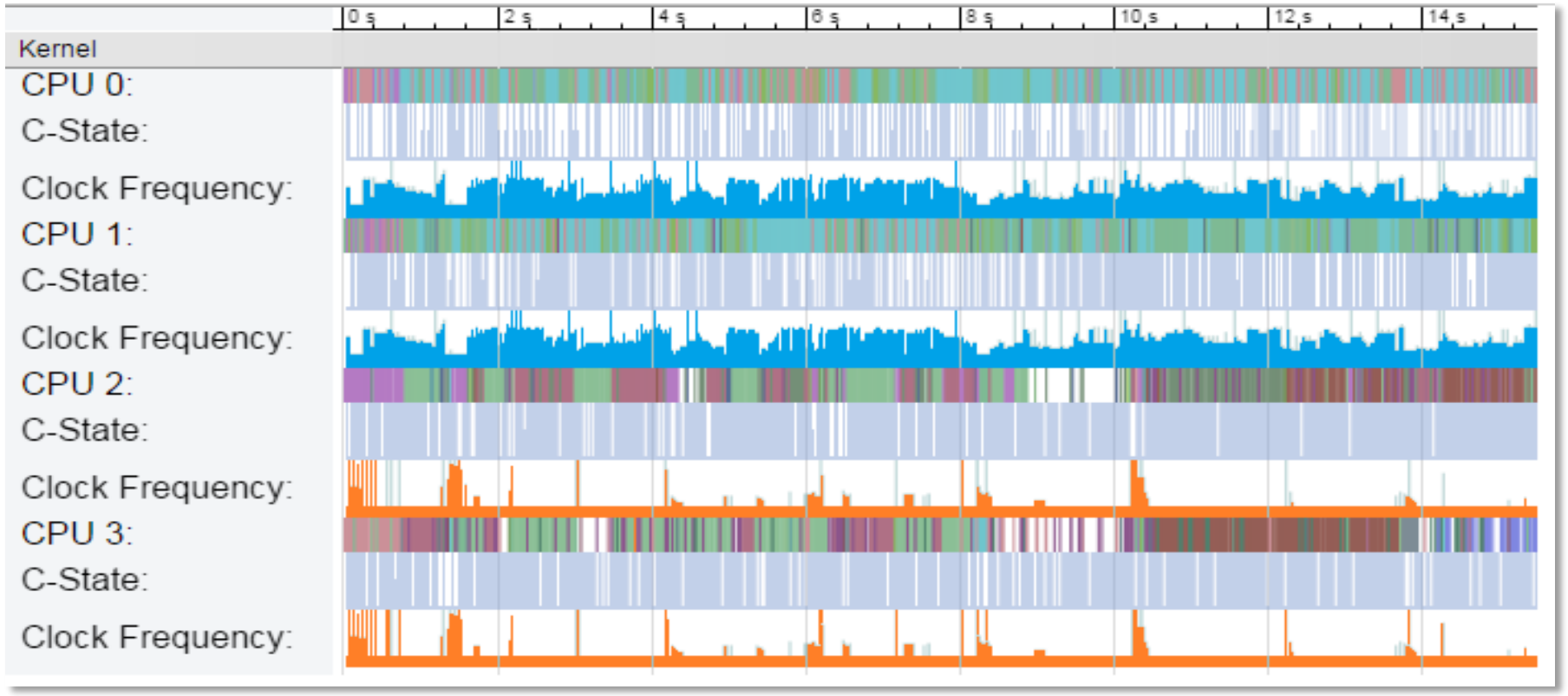


- Cluster A and Cluster B consist of CPUs that are equally capable (same IPC).
- Each cluster has a different FMAX.
- Power and performance curves for each upto the FMAX of cluster B are very similar.

Spreading Tasks

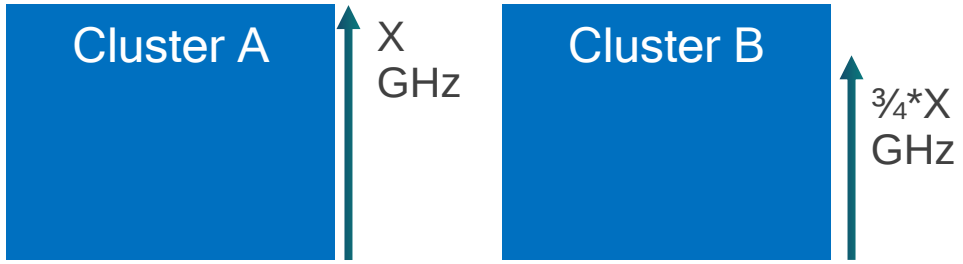


- Current EAS wake-up placement algorithm will always pack on little cluster until overutilization point
  - This harms both performance and power on multicluster SMP
  - Example real world use case: Play TempleRun!



EAS+PELT vs EAS+WinLT

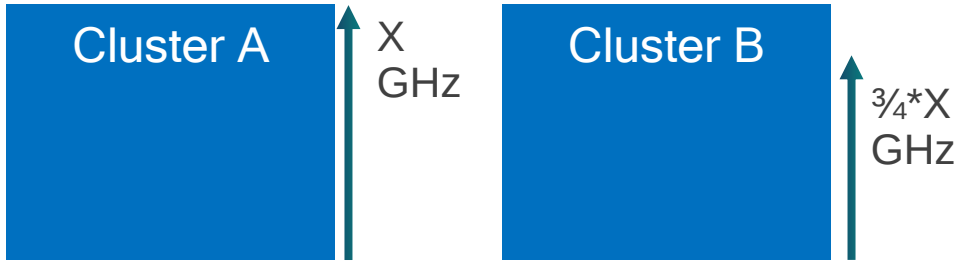
Benchmark Testing



- Single 3.18 based kernel with both load tracking mechanisms on SMP multi-cluster architecture
- EAS v5
- Same governor used for all testing: sched-freq
- No sched-tune boosting
- With WinLT, the only change is that prev_runnable_sum is used instead of util_avg to determine cpu usage when setting CPU frequency.
  - A better policy, one similar to the window stats policy of max(avg, recent) will be investigated.
- Placement with WinLT to be investigated

EAS+PELT vs EAS+WinLT

Benchmark Results



Test	Result
PCMark Browser	15% improvement with WinLT for comparable power
PCMark Photo Editing	10% improvement with WinLT for comparable power
PCMark Video	Same
Antutu	Same
Geekbench	Same
TempleRun	EAS+PELT requires a use-case specific 5% sched-tune boost in order to hit the same performance (FPS) as EAS+WinLT.

Conclusions

- WinLT is a proven load tracking mechanism that is running in most Qualcomm® Snapdragon™-based multi-cluster Android phones released over the past two years.
- Benchmark results indicate performance and power improvements in real world usecases when WinLT statistics are used to guide frequency settings.
- EAS+WinLT is experimental at this stage. Commercial quality tuning may provide better results.
- EAS+WinLT will likely require less schedtune-style boost tuning due to increased responsiveness
- More investigation is required – placement using WinLT based task load tracking, and perhaps modifications to PELT to address the concerns highlighted in this presentation.

Thank you

Follow us on:

For more information, visit us at:
www.qualcomm.com & www.qualcomm.com/blog