Improving PELT

Decay Clamping vs Utilization Estimation

Morten Rasmussen, Patrick Bellasi
<morten.rasmussen@arm.com> <patrick.bellasi@arm.com>

Agenda

- Why PELT is not “good enough”
- Decay Clamping
- Utilization Estimation

ARM

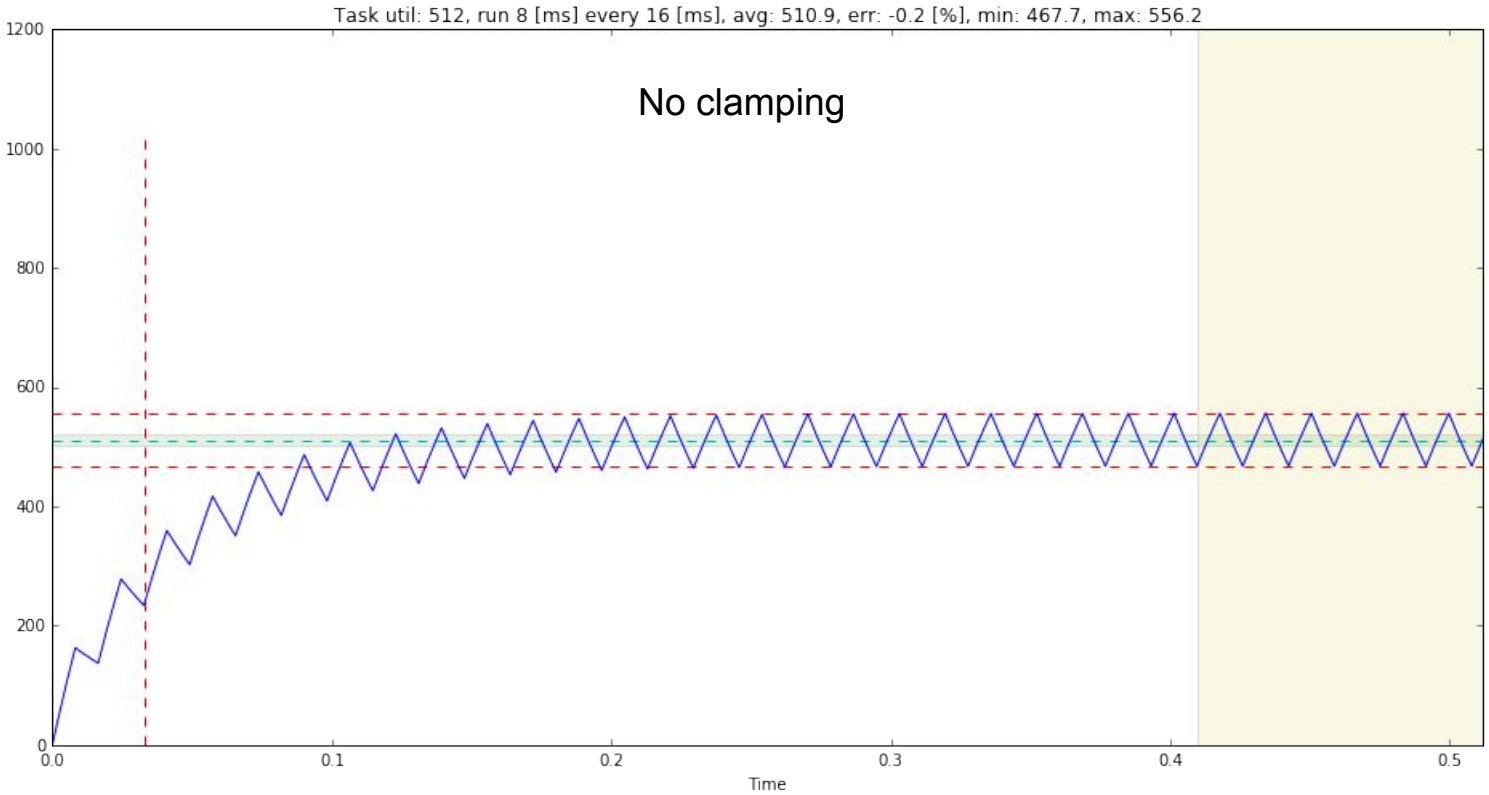
Decay Clamping

- **Problem:** Tasks with long sleep periods loose too much of their accumulated utilization leading to wrong utilization estimates.

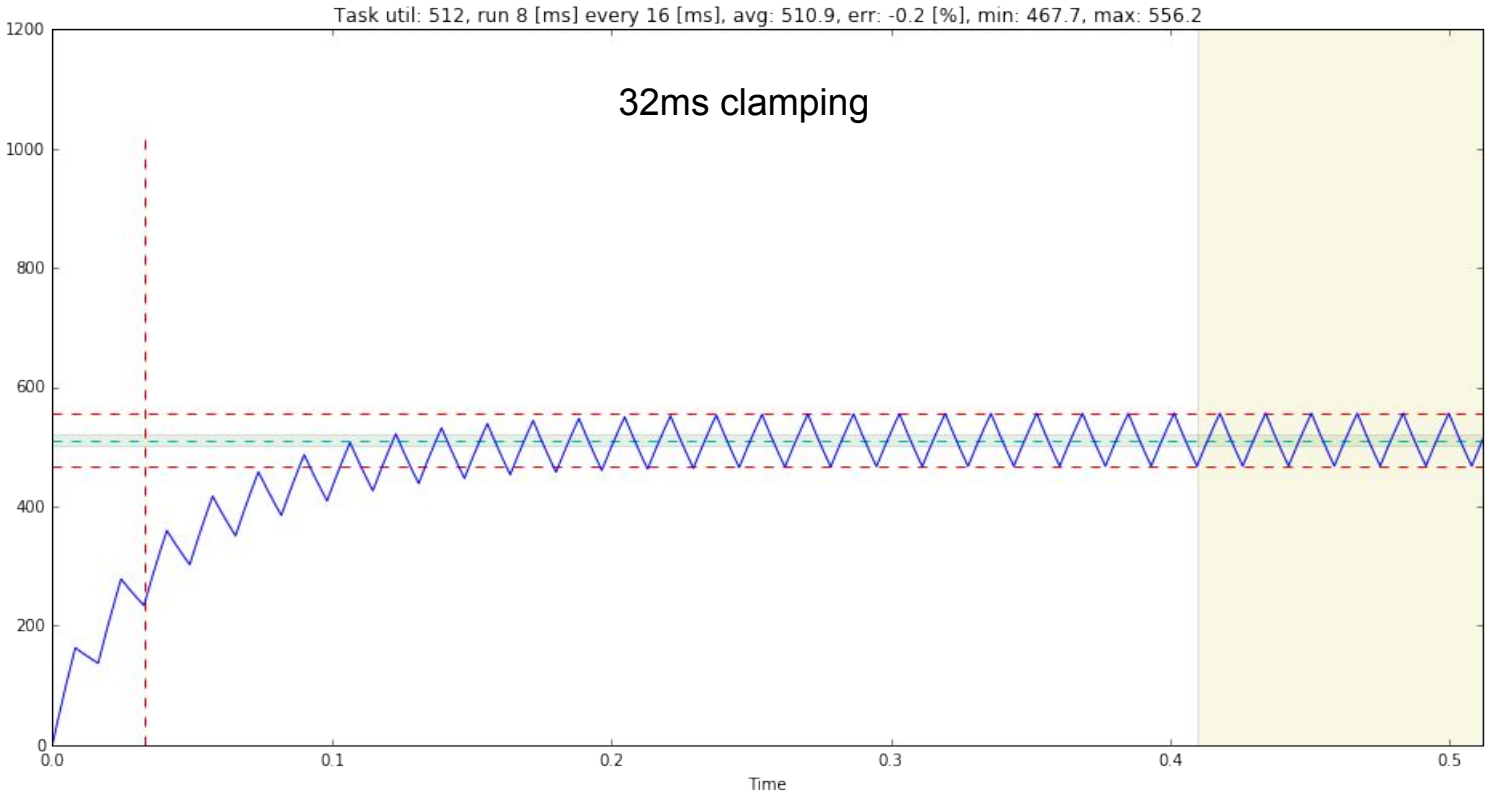
- **Proposal:** Ignore sleep time beyond a fixed threshold, essentially clamping the utilization decay at wake-up.

- Discussed at LPC 2016. RFC proof-of-concept for evaluation is ready.

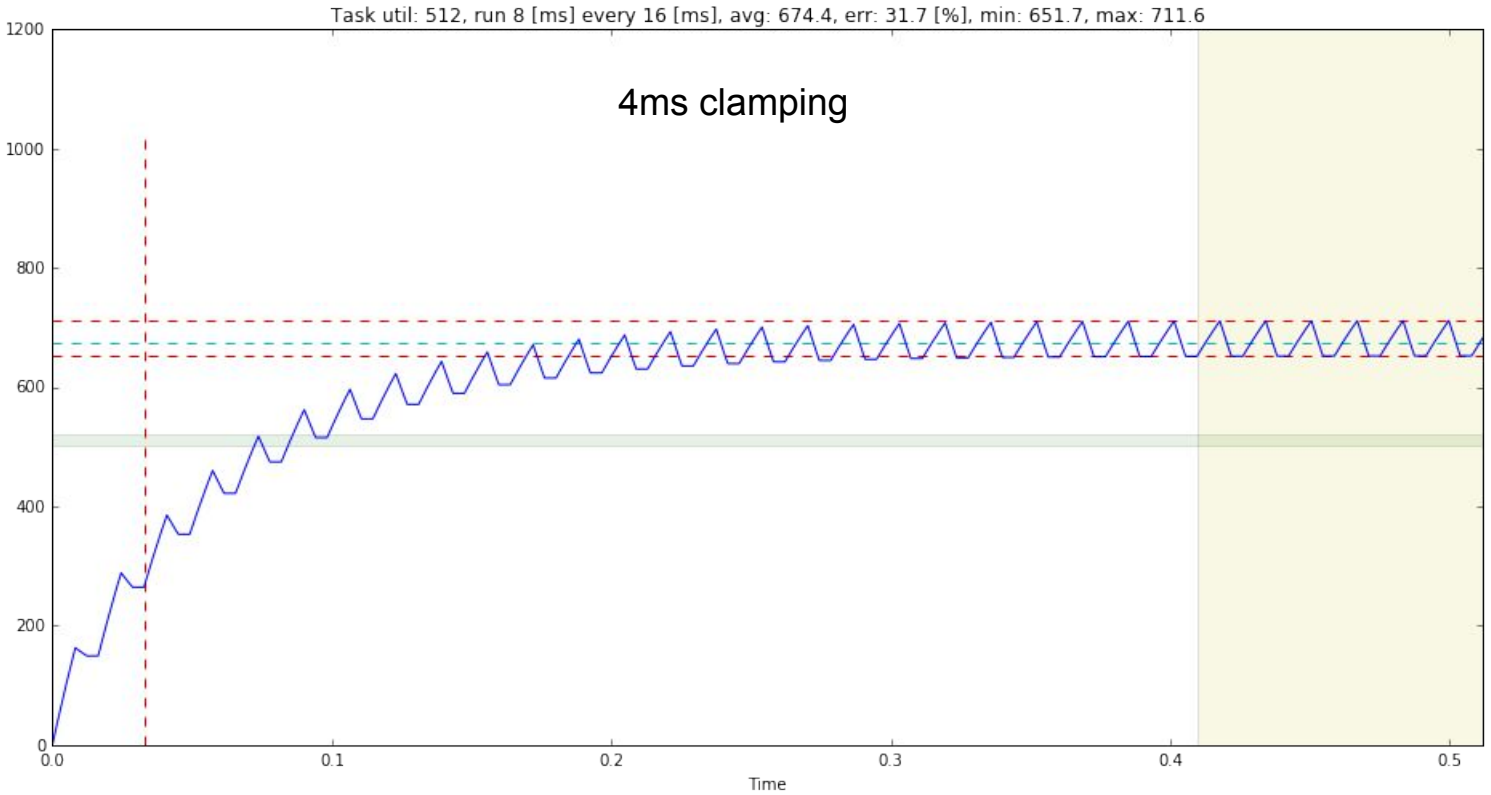
Analysis Decay Clamping: Android-like periodic task



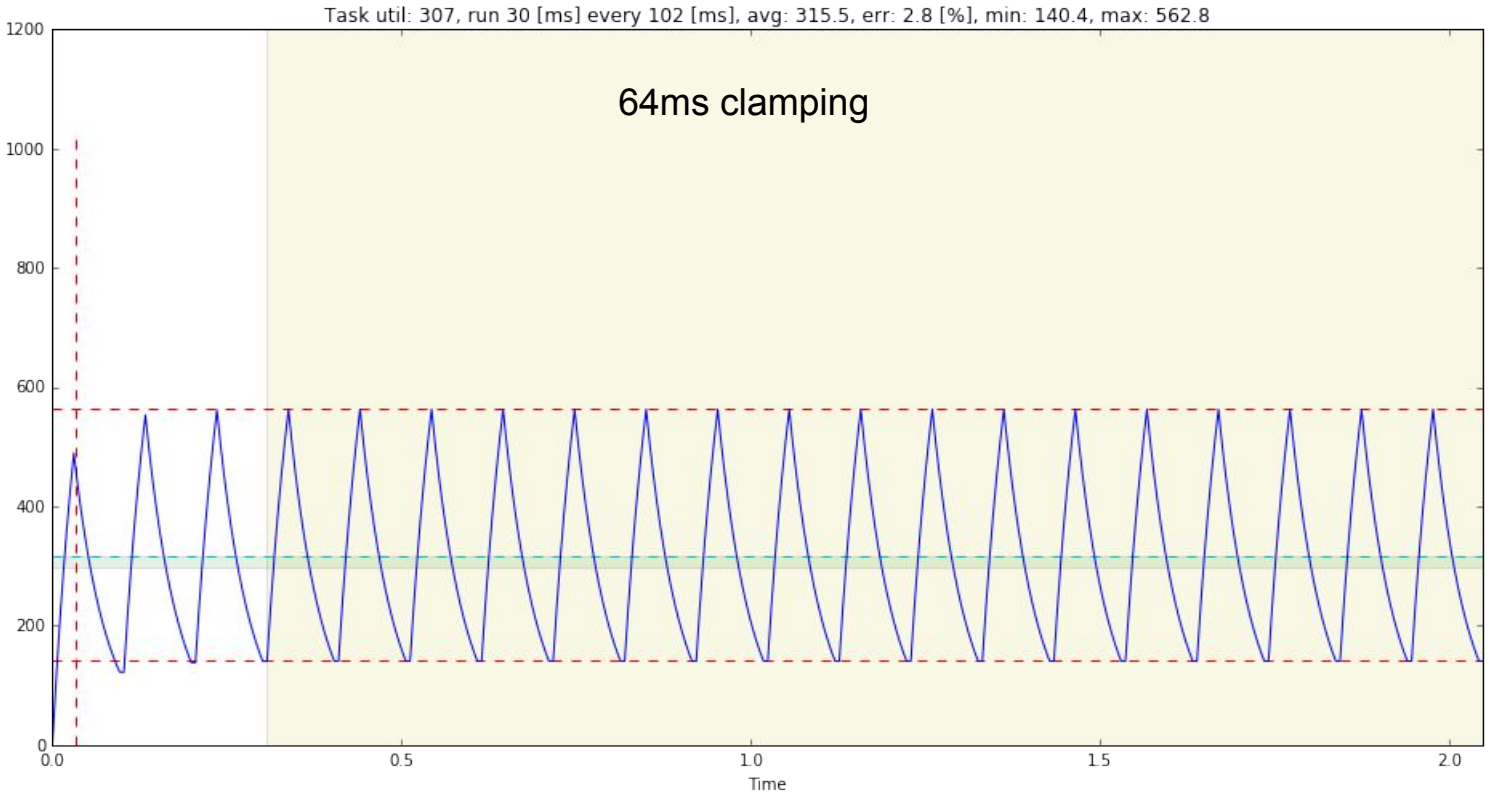
Analysis Decay Clamping: Android-like periodic task



Analysis Decay Clamping: Android-like periodic task

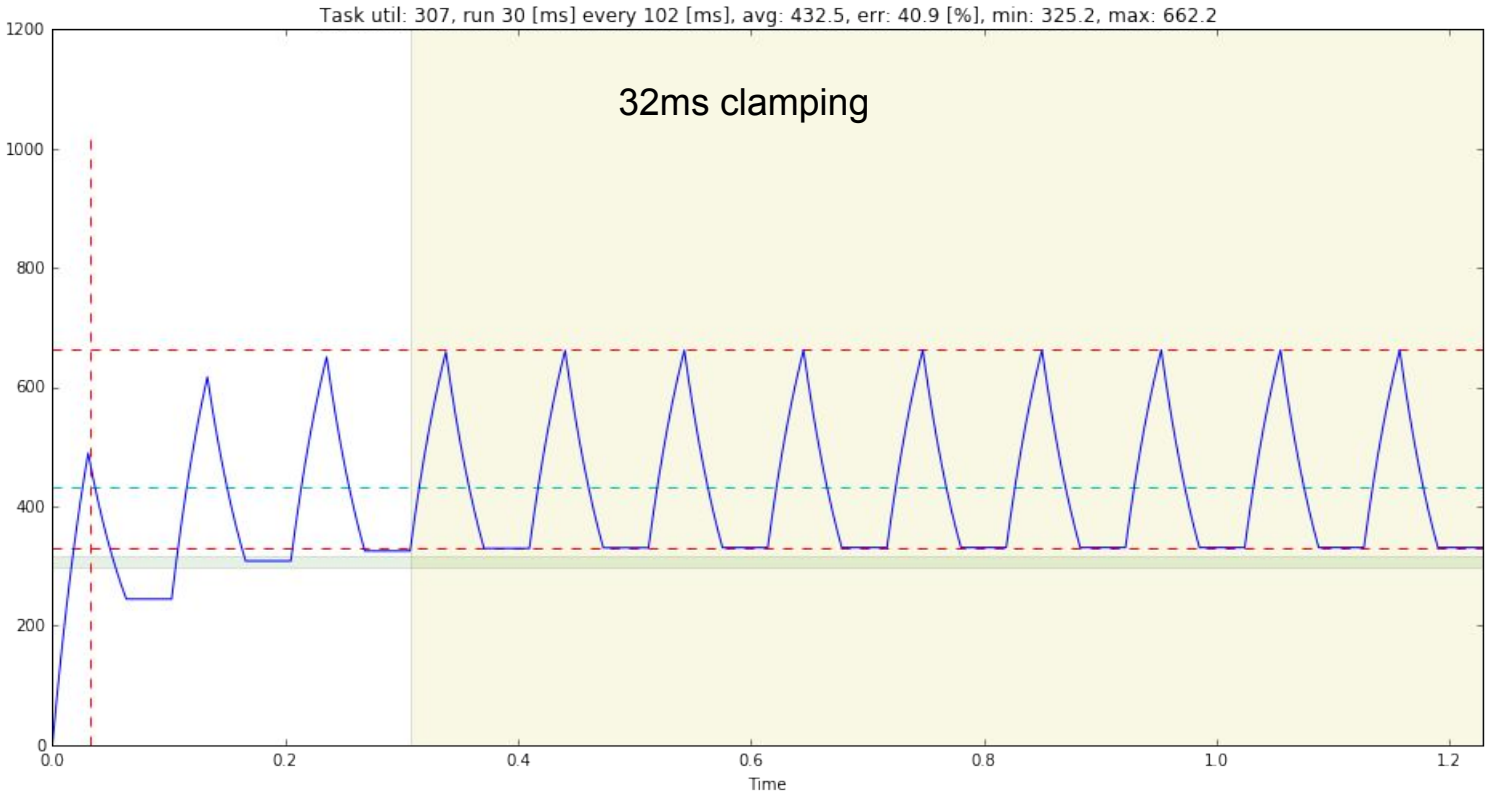


Analysis Decay Clamping: Long period task



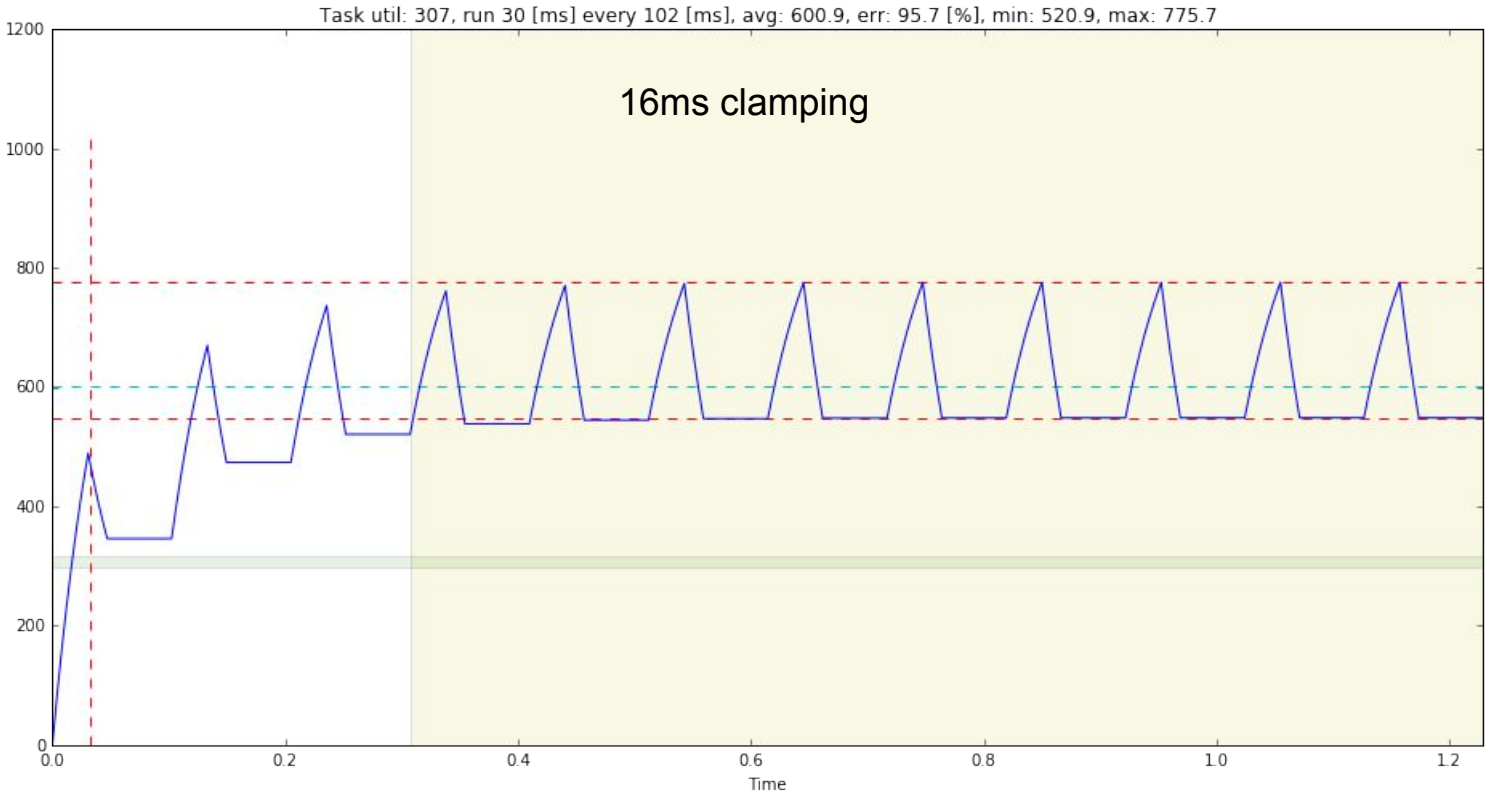
ARM

Analysis Decay Clamping: Long period task



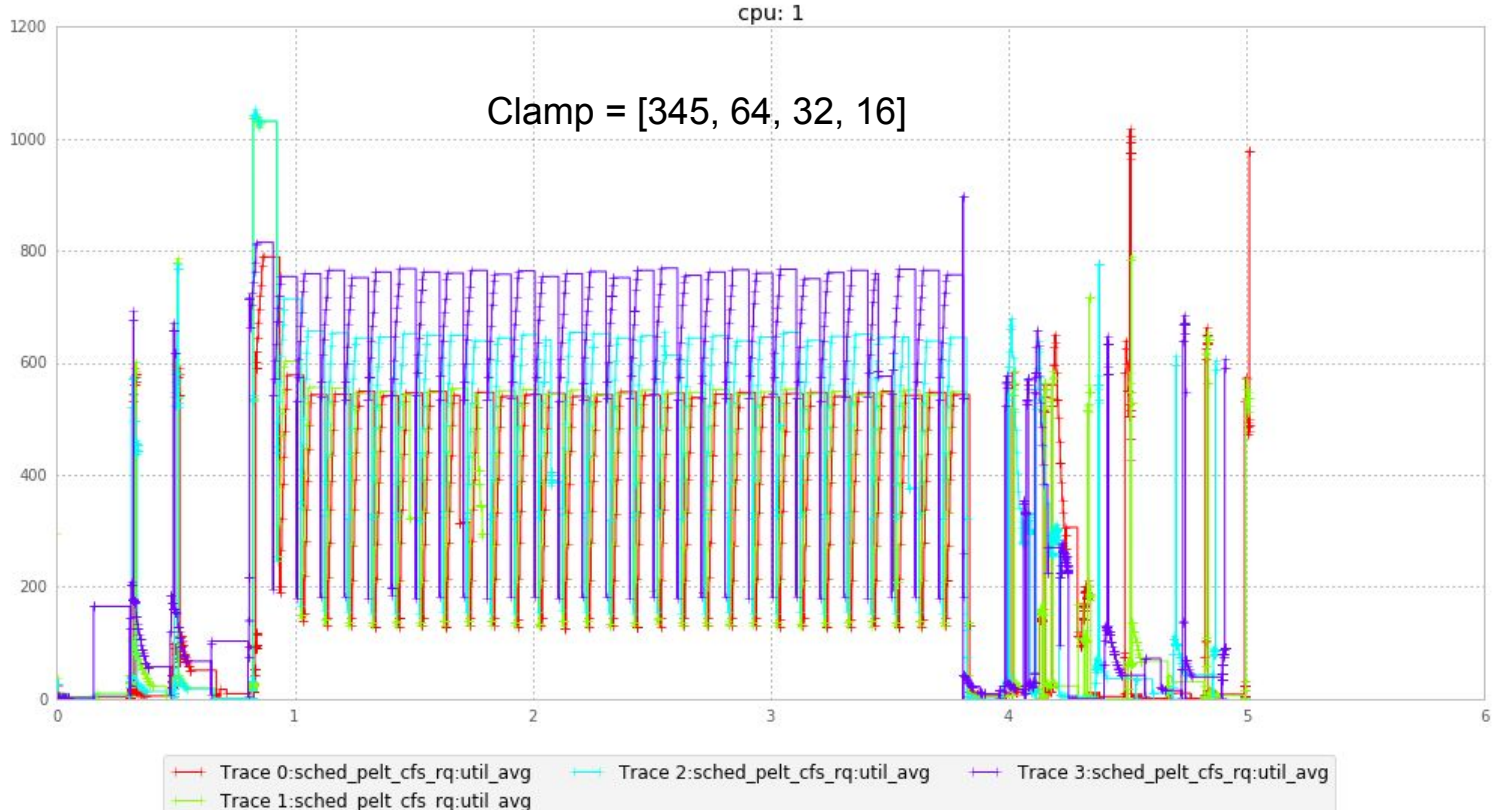
ARM

Analysis Decay Clamping: Long period task



ARM

Proto-type: Long period task traced



ARM

Proto-type: Long period task schedutil performance

performance

clamp	min	max	mean
345	30464	30742	30581
64	30515	30667	30582
32	30507	30811	30582
16	30501	30731	30578
util_est	30515	30809	30604

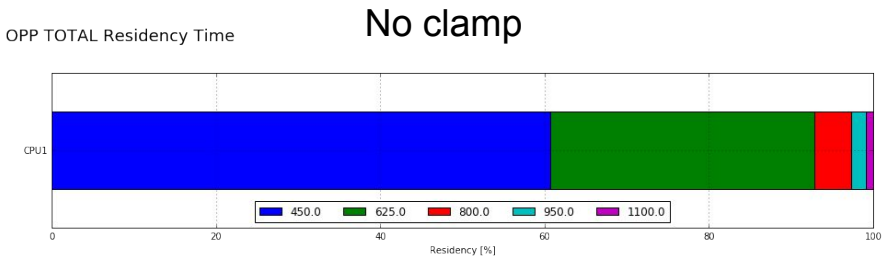
schedutil

clamp	min	max	mean
345	42573	70093	66958
64	36402	68650	66787
32	32845	68774	64914
16	37921	50341	48603
util_est	34736	45223	44122

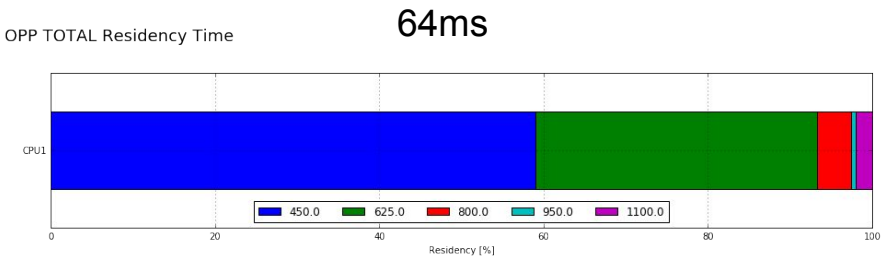
ARM

Proto-type: Long period task schedutil performance

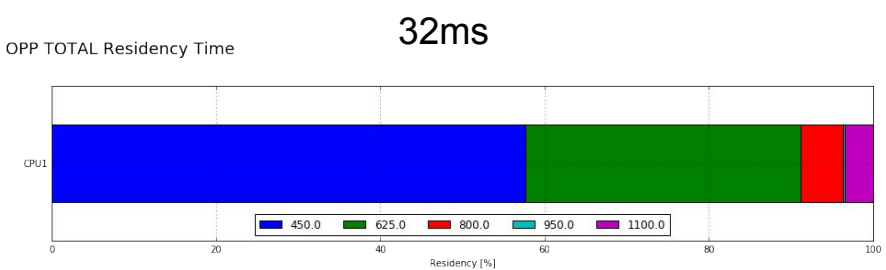
No clamp



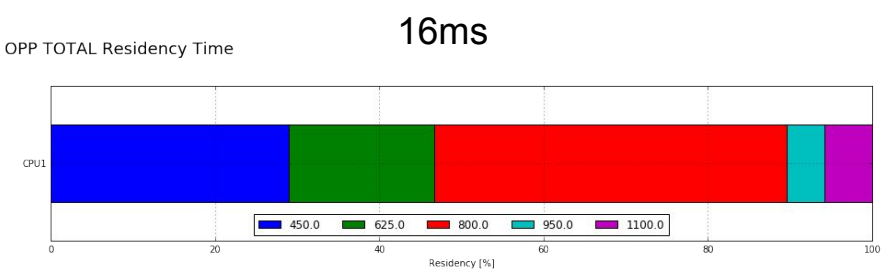
64ms



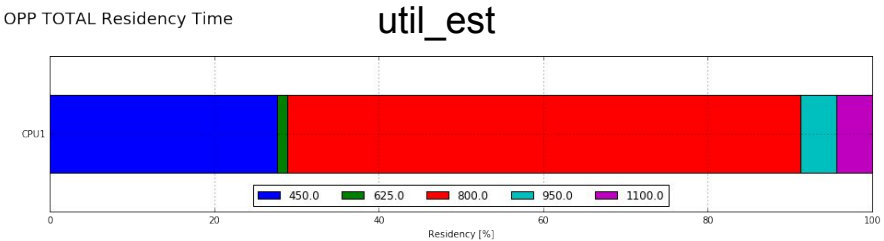
32ms



16ms



util_est

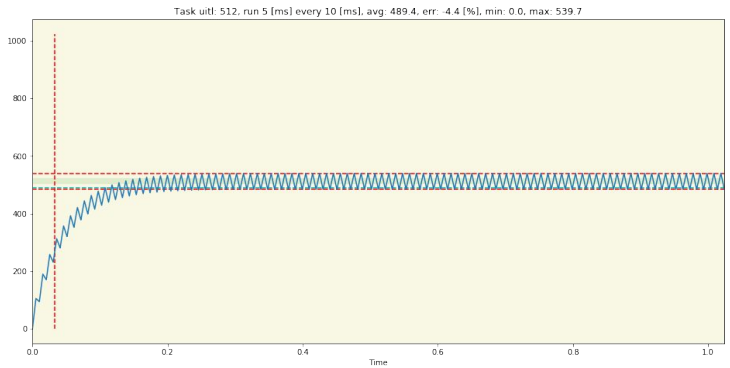


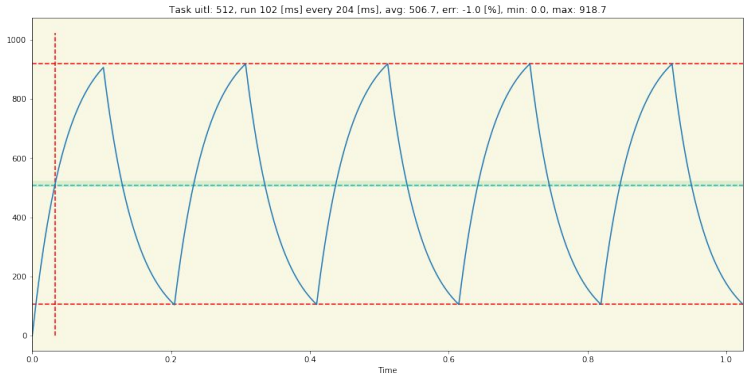
ARM

PELT: Why is not good enough?

It has “fast dynamic”: it’s updated “every” time the scheduler has an opportunity

- makes somehow “instantly outdated” every decision we take





- it does not “consolidate information” about previous activations

It’s “slow”: a task waking-up after a long sleep has a small utilization (once enqueued)

- it takes tens of milliseconds to represent the CPU demand of that task

ARM

Utilization Estimation: Fundamental Idea

Add an **aggregator** on top of the PELT **estimator**

- keep track of what "we learned" about task's previous activations
- generate a "new" signal on top of PELT

Build a **low-overhead statistic** for SEs and CPUs

- Tasks, at dequeue_task_fair() time
- Root RQs, at {dequeue,enqueue}_task_fair time

  since we are interested mainly on OPP selection

Use **getter methods** to define which signal to use

- {task,cpu}_util_est()

  Tasks: max(util_avg, util_est.ewma, util_est.last)

  CPUs: max(util_avg, util_est.last)

```
@@ -4834,6 +4841,14 @@ enqueue_task_fair(struct rq *rq, struct task_struct *p, int flags)
        if (!se)
                add_nr_running(rq, 1);

+       /*
+        * Update (top level CFS) RQ estimated utilization.
+        * NOTE: the following code assume that we never change the
+        *       utilization estimation policy at run-time.
+        */
+       cfs_rq = &(task_rq(p)->cfs);
+       cfs_rq->avg.util_est.last += task_util_est(p);
+
        hrtick_update(rq);
 }

@@ -4893,6 +4908,24 @@ static void dequeue_task_fair(struct rq *rq, struct task_struct *p, int
        if (!se)
                sub_nr_running(rq, 1);

+       /*
+        * Update (top level CFS) RQ estimated utilization
+        * NOTE: for RQs we alwasy use util_est.last since we do not track an
+        *       EWMA, which is tracked only for Tasks.
+        */
+       cfs_rq = &(task_rq(p)->cfs);
+       cfs_rq->avg.util_est.last = max_t(long,
+                                         cfs_rq->avg.util_est.last - task_util_est(p), 0);
+
+       /* Update Task's estimated utilization */
+       if (task_sleep) {
+               /* Keep track of the utilization for the last activation */
+               p->se.avg.util_est.last = task_util(p);
+               /* Update EWMA for Task utilization */
+               ewma_util_add(&p->se.avg.util_ewma, task_util(p));
+               p->se.avg.util_est.ewma = ewma_util_read(&p->se.avg.util_ewma);
+       }
+
        hrtick_update(rq);
 }
```

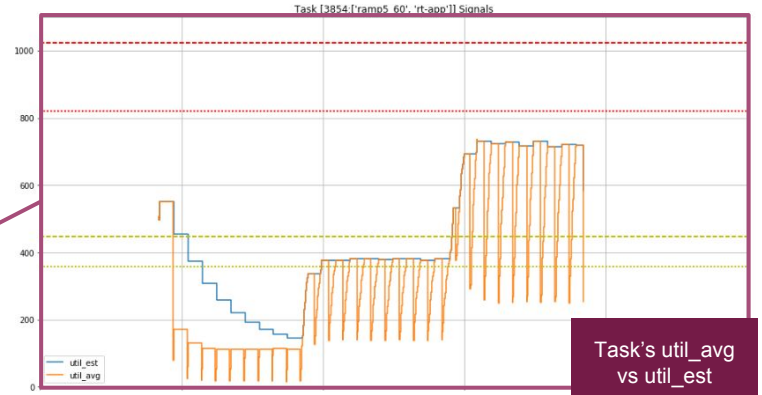
Utilization Estimation: Initial Proposal

Patches going to be posted on LKML

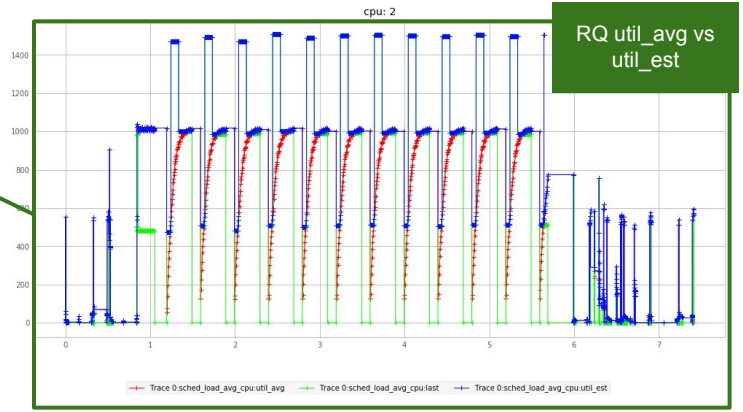
- git://www.linux-arm.org/linux-pb.git eas/pelt/utilest

Evaluation consisting of synthetic workloads (with[1] and without[2] utilest)

- Periodic (60% every 300ms)
- Ramp (5,25,45% every 100ms)
- Two tasks co-scheduled (50% every 400ms)
- Fake “render thread” (60% every 16ms)
- Migrating task (20% every 20ms)



Task’s util_avg vs util_est



RQ util_avg vs util_est

[1] https://gist.github.com/derkling/0d07b97ca18cc5eac25e51404e81169f
[2] https://gist.github.com/derkling/e1cfd776d310365528010563fb24b06a

ARM

Utilization Estimation: Possible Future Extensions

A **per-task policy** can be used to **select the estimation** signal to be used, e.g.

- **“boosted tasks”** starts from **max(ewma, last)**
- **“background tasks”** always starts from the **decayed util_avg**

Experiment by **tracking other metrics**, instead of max currently aggregated

- we can experiment by tracking other metrics

  e.g. (max-min)/2?

- can the util_est be used to “compensate” for stale utilization on idle CPUs

  e.g. return a “virtually decayed” utilization on-demand (i.e. when we need to look at an idle CPU
  NOTE: goal is to drive OPPs and tasks placement, thus perhaps it’s just enought to track top level RQs

Backup Slides