Errors in PELT signal due to period boundary

Joel Fernandes <joelaf@google.com>

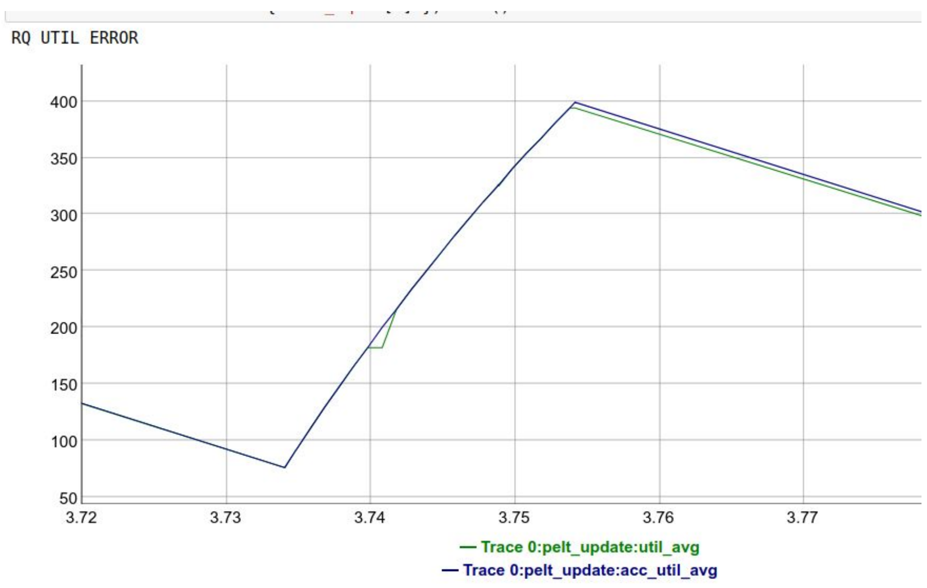
Introduction

The PELT signal (sa->load_avg and sa->util_avg) are not updated if the amount accumulated during a single update doesn't cross a period boundary. This is fine in cases where the amount accrued is much smaller than the size of a single PELT window (1ms) however if the amount accrued is high then the error (calculated against what the actual signal would be had we updated the averages) can be quite high - as much 3-6% or more in my testing. On plotting waveforms of the signals, I found that there are noticeable glitches in the waveform that could have been avoided had we considered that the accrued amount is high enough that the sum and averages have diverged. Other than glitches, I also see that the signal is slightly lower on many occasions than it could have been.

Summary of issues

* GLITCH in signal

At 3.74s, there is a 9% error in util_avg (200 vs 182) - or 1.8% absolute error (measured against the maximum util of 1024) - this causes a glitch and makes the signal less smooth.
Legend: GREEN is actual signal, BLUE is the corrected signal.
X-axis is time, Y-axis is util_avg signal value for the RQ

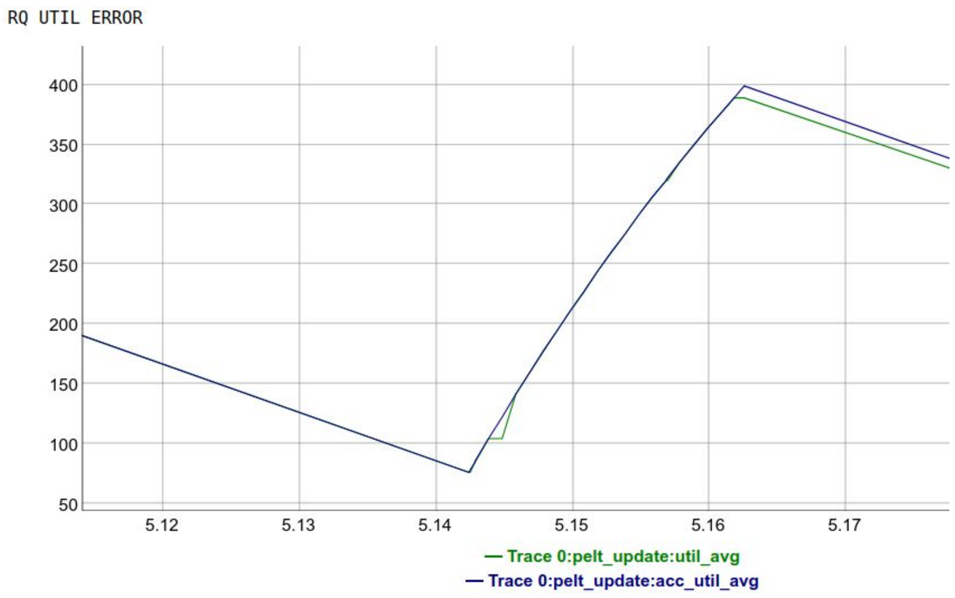


```
      thread0-1050  [001]     9.233629: bprint:              task_tick_fair: pelt_update:
util_avg=164 load_avg=165 acc_load_avg=165 acc_util_avg=164 util_err=0 load_err=0
load_sum=7864850 sum_err=0 delta_us=975 cfs_rq=0 ret=1
         thread0-1050  [001]     9.233629: bprint:              task_tick_fair: pelt_update:
util_avg=165 load_avg=165 acc_load_avg=165 acc_util_avg=165 util_err=0 load_err=0
load_sum=7758917 sum_err=0 delta_us=975 cfs_rq=1 ret=1
         thread0-1050  [001]     9.234628: bprint:              task_tick_fair: pelt_update:
util_avg=182 load_avg=182 acc_load_avg=182 acc_util_avg=182 util_err=0 load_err=0
load_sum=8692674 sum_err=0 delta_us=977 cfs_rq=0 ret=1
         thread0-1050  [001]     9.234629: bprint:              task_tick_fair: pelt_update:
util_avg=182 load_avg=183 acc_load_avg=183 acc_util_avg=182 util_err=0 load_err=0
load_sum=8571603 sum_err=0 delta_us=977 cfs_rq=1 ret=1
         thread0-1050  [001]     9.235635: bprint:              task_tick_fair: pelt_update:
util_avg=199 load_avg=200 acc_load_avg=200 acc_util_avg=199 util_err=0 load_err=0
load_sum=9506855 sum_err=0 delta_us=982 cfs_rq=0 ret=1
         thread0-1050  [001]     9.235636: bprint:              task_tick_fair: pelt_update:
util_avg=182 load_avg=183 acc_load_avg=200 acc_util_avg=200 util_err=18 load_err=17
load_sum=9577171 sum_err=-1005568 delta_us=982 cfs_rq=1 ret=0

         thread0-1050  [001]     9.236630: bprint:              task_tick_fair: pelt_update:
util_avg=216 load_avg=216 acc_load_avg=216 acc_util_avg=216 util_err=0 load_err=0
load_sum=10292326 sum_err=0 delta_us=972 cfs_rq=0 ret=1
```

* Signal doesn’t peak as much as it normally does.

At 5.16s, there is a 2.5% error in util_avg during the peak (389 vs 399) or 1% absolute error (measured against maximum util of 1024) causing lowered peak of util_avg with delta ~720us. This due to the dequeue that happens before the end of period boundary, see traces below:
Legend: GREEN is actual signal, BLUE is the corrected signal.
X-axis is time, Y-axis is util_avg signal value for the RQ



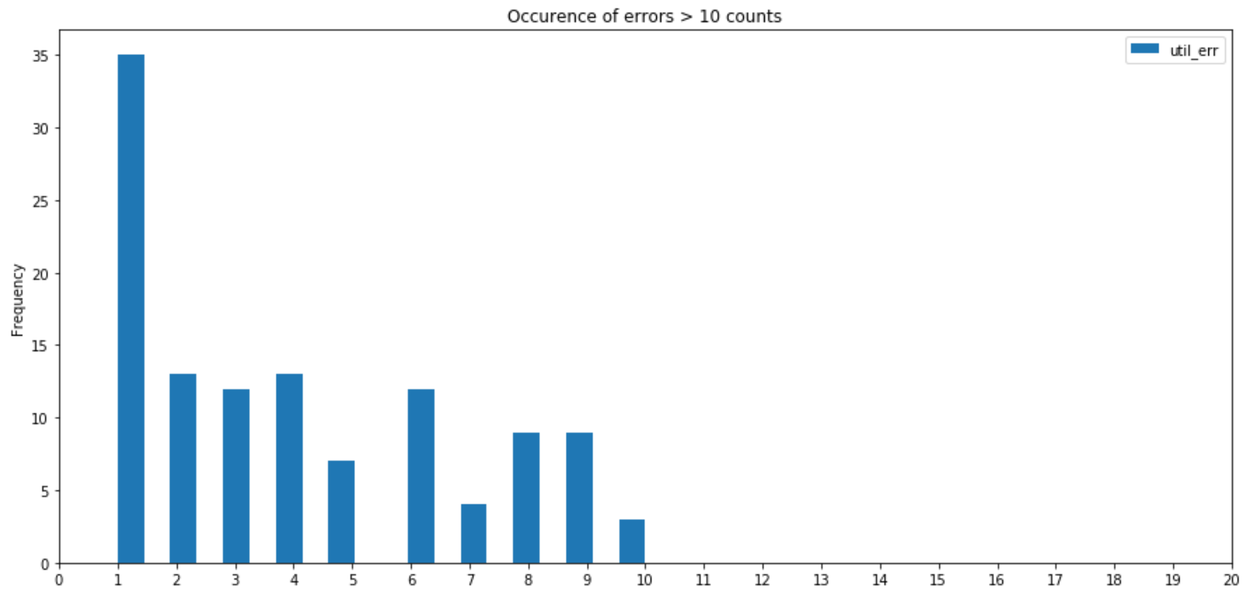
Traces for the “reduced peak issue”

```
        thread0-1050  [001]   10.654690: bprint:              task_tick_fair: pelt_update:
util_avg=363 load_avg=426 acc_load_avg=426 acc_util_avg=363 util_err=0 load_err=0
load_sum=20029072 sum_err=0 delta_us=977 cfs_rq=1 ret=1
        thread0-1050  [001]   10.655689: bprint:              task_tick_fair: pelt_update:
util_avg=373 load_avg=377 acc_load_avg=377 acc_util_avg=373 util_err=0 load_err=0
load_sum=17656717 sum_err=0 delta_us=978 cfs_rq=0 ret=1
        thread0-1050  [001]   10.655691: bprint:              task_tick_fair: pelt_update:
util_avg=376 load_avg=438 acc_load_avg=438 acc_util_avg=376 util_err=0 load_err=0
load_sum=20584978 sum_err=0 delta_us=978 cfs_rq=1 ret=1
        thread0-1050  [001]   10.656681: bprint:              task_tick_fair: pelt_update:
util_avg=373 load_avg=377 acc_load_avg=390 acc_util_avg=386 util_err=13 load_err=13
load_sum=18651021 sum_err=-994304 delta_us=971 cfs_rq=0 ret=0
        thread0-1050  [001]   10.656683: bprint:              task_tick_fair: pelt_update:
util_avg=389 load_avg=450 acc_load_avg=450 acc_util_avg=389 util_err=0 load_err=0
load_sum=21120780 sum_err=0 delta_us=971 cfs_rq=1 ret=1
        thread0-1050  [001]   10.657420: bprint:              dequeue_task_fair: pelt_update:
util_avg=396 load_avg=400 acc_load_avg=400 acc_util_avg=396 util_err=0 load_err=0
load_sum=18987624 sum_err=0 delta_us=720 cfs_rq=0 ret=1
        thread0-1050  [001]   10.657422: bprint:              dequeue_task_fair: pelt_update:
util_avg=389 load_avg=450 acc_load_avg=458 acc_util_avg=399 util_err=10 load_err=8
load_sum=21858060 sum_err=-737280 delta_us=720 cfs_rq=1 ret=0
```

After the following patch, errors are now bounded to maxium of 10 (~1% absolute)

```
 @@ -3201,9 +3202,12 @@ ___update_load_sum(u64 now, int cpu, struct sched_avg *sa,
         * accrues by two steps:
         *
         * Step 1: accumulate *_sum since last_update_time. If we haven't
-        * crossed period boundaries, finish.
+        * crossed period boundaries and the time since last update is small
+        * enough, we're done.
         */
-       if (!accumulate_sum(delta, cpu, sa, load, runnable, running))
+       periods = accumulate_sum(delta, cpu, sa, load, runnable, running);
+
+       if (!periods && delta < 512)
                return 0;

        return 1;
```



Performance is still as good (with a little bit of overhead):

Unixbench shell 8 tests,

Score Without:
9157.9
9134.6

Score With
9103.2
9082

Hackbench: total time for 'hackbench-perf' LKP test
With: 183.03
Without: 183.12