PELT Load-Tracking Problem While Running Geekbench 3.0

Geekbench is a widely used benchmark on Android and iOs devices. It's also available on Linux and Mac OS X. It somehow provides a cross-platform benchmark which we can use as an index of how powerful the tested CPUs are?

Geekbench consists of 3 sections of tests. It has a web page describing what the three sections and the benchmarks in each section are. See http://support.primatelabs.com/kb/geekbench/interpreting-geekbench-3-scores (http://support.primatelabs.com/kb/geekbench/interpreting-geekbench-3-scores). For each test of Geekbench 3, both single thread and multi-thread versions are run. For the multi-thread versions, the same contents are repeated 4 times. There is a synchronization point between each iteration.

Below, we use a log of full-run of geekbench on recorded by 'trace-cmd record -e 'sched' -e 'power''. The log is placed in a directory named "eas-sched-dvfs-fork-migration-all".

First, we plot a kernelshark-like events figure with ARM's Trappy. From the figure, we can see Geekbench running with different single-thread and multi-thread tests. An example single-thread and multi-thread test consists of tasks with pid 1274-1278 (1274: single thread, 1275-1275: multiple (4) threads)

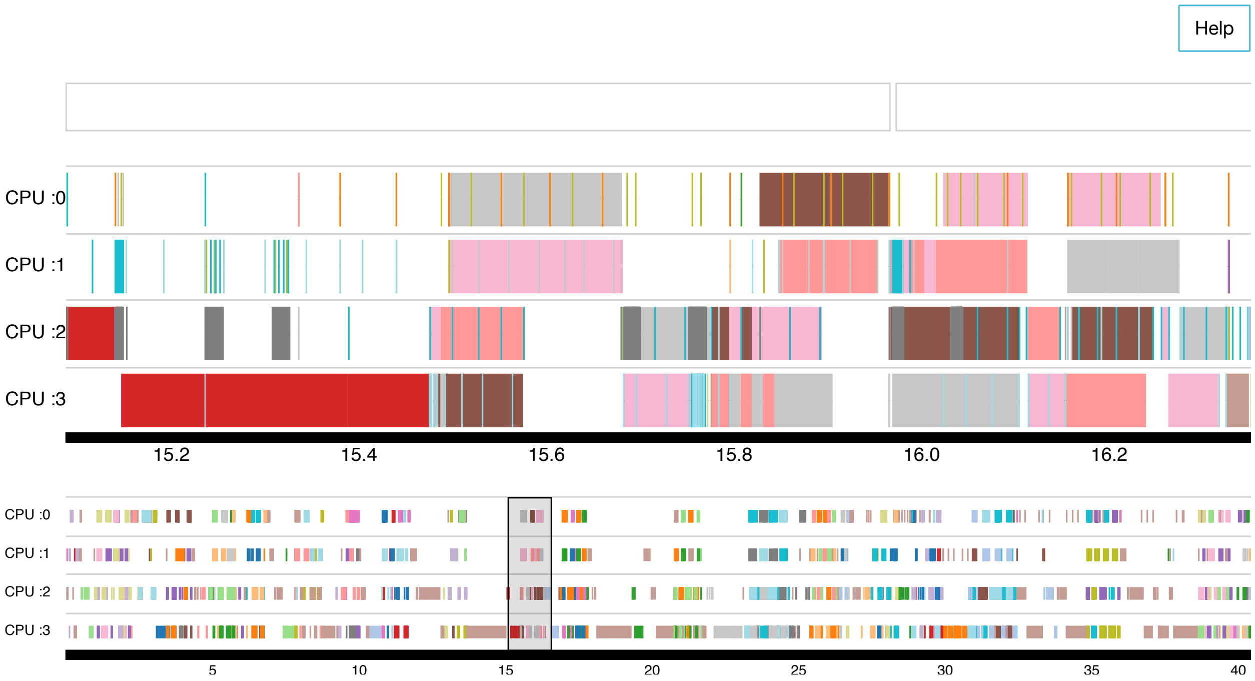
In [1]:

```
%pylab inline
import sys,os
sys.path.append("..")
import tempfile
import shutil
import trappy
import trappy.plotter.Utils
import trappy.plotter.EventPlot as EventPlot

trace_sched = "eas-sched-dvfs-fork-migration-all/trace.dat"

def setup_sched():
    tDir = tempfile.mkdtemp(dir="/tmp", prefix="trappy_doc", suffix = ".tempDir")
    shutil.copyfile(trace_sched, os.path.join(tDir, "trace.dat"))
    return tDir
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:

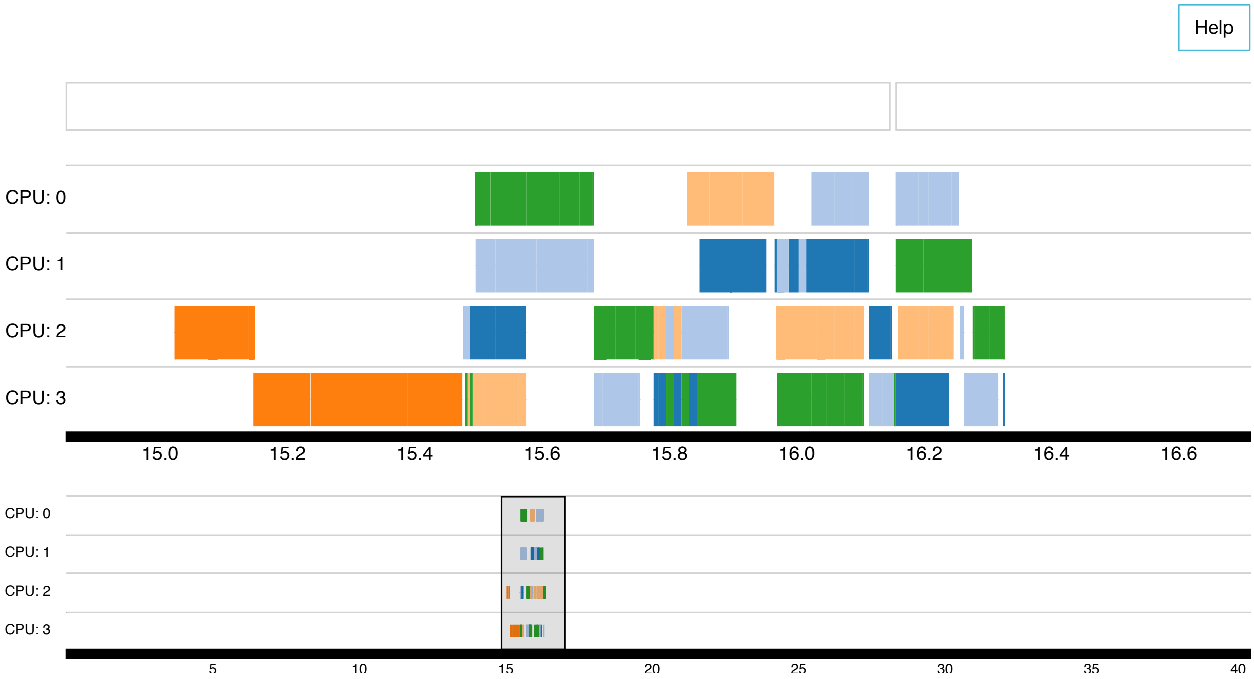
```
f = setup_sched()
trappy.plotter.plot_trace(f)
```



Ideally, to get better benchmark scores, for each test, we should have a figure like what pid 1274-1278 do. Or, a little bit better, for each iteration of multi-thread test, thread should start and end at roughly the same time. To achive this, some kinds of randomization and migration should be forced upon the four threads. For current CFS or EAS-enhanced CFS, it seems the way for the scheduler to get the best results is to start all the four threads at the same time, then the two thread in big cluster will be finished earlier, and the two threads on the LITTLE cluster consequently will be moved to the big cluster.

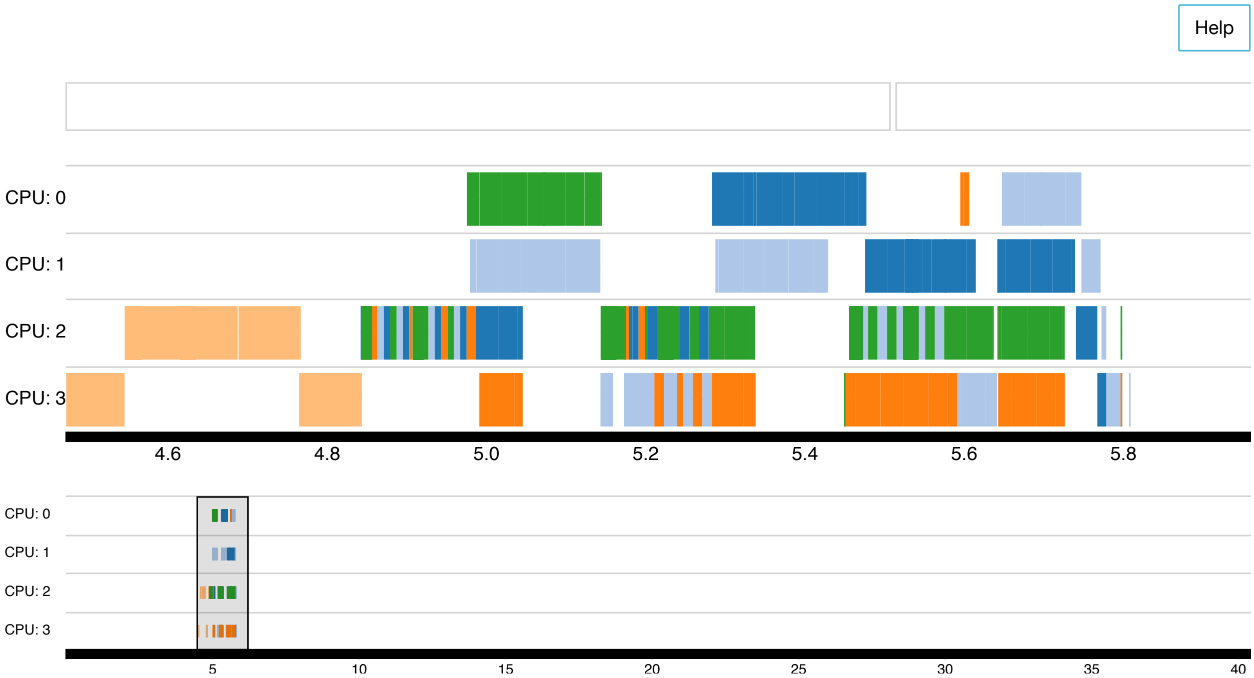
In [3]:
```
trace_sched_dir = "eas-sched-dvfs-fork-migration-all"
run = trappy.Run(trace_sched_dir)

def trace_events(execnames, pids):
    data, procs, domain = trappy.plotter.Utils.get_trace_event_data(run, execnames, pids)
    trace_graph = EventPlot.EventPlot(data, procs, domain, lane_prefix="CPU: ", num_lanes=int(run._c
    return trace_graph
```

In [4]:
```
l = trace_events(None, [1274, 1275, 1276, 1277, 1278])
l.view()
```



Unfortunately, most tests in Geekbench work like pid 1244 - 1248. In the beginning of multi-threaded iterations (around 4.8483736 to 4.980974 second), pid 1245-1248 all contend for CPU2. Or worse.

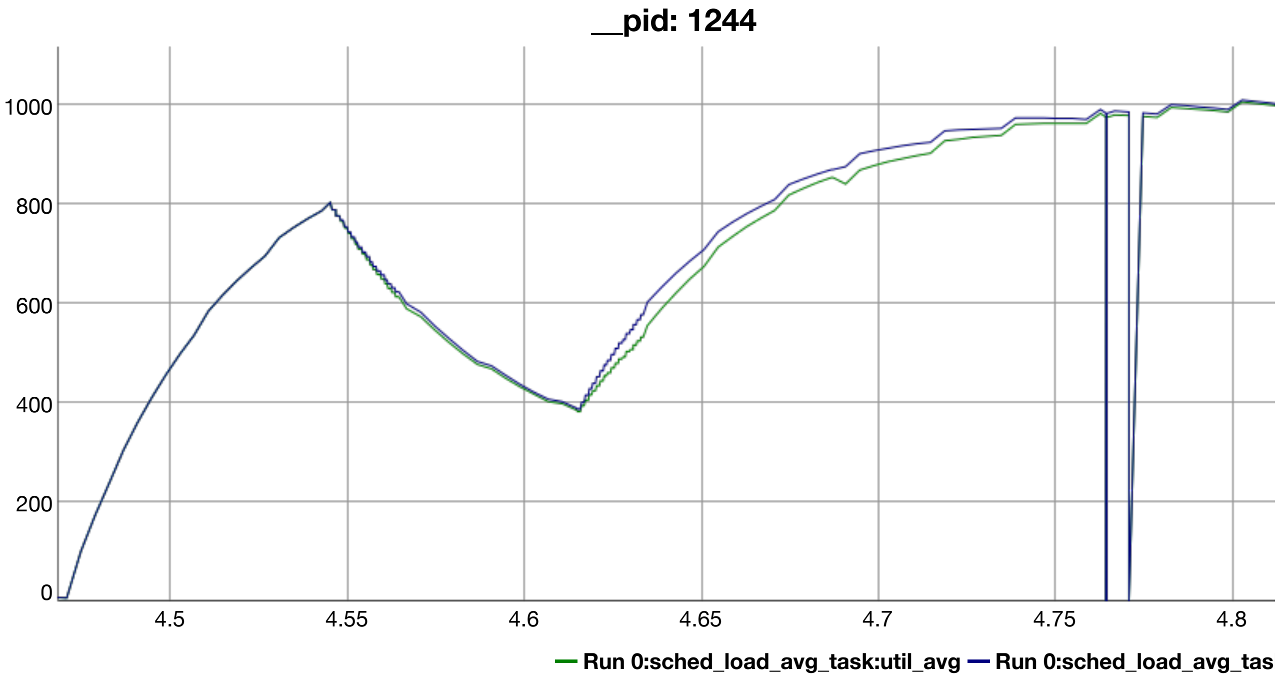
In [6]:
```
l = trace_events(None, [1244, 1245, 1246, 1247, 1248])
l.view()
```

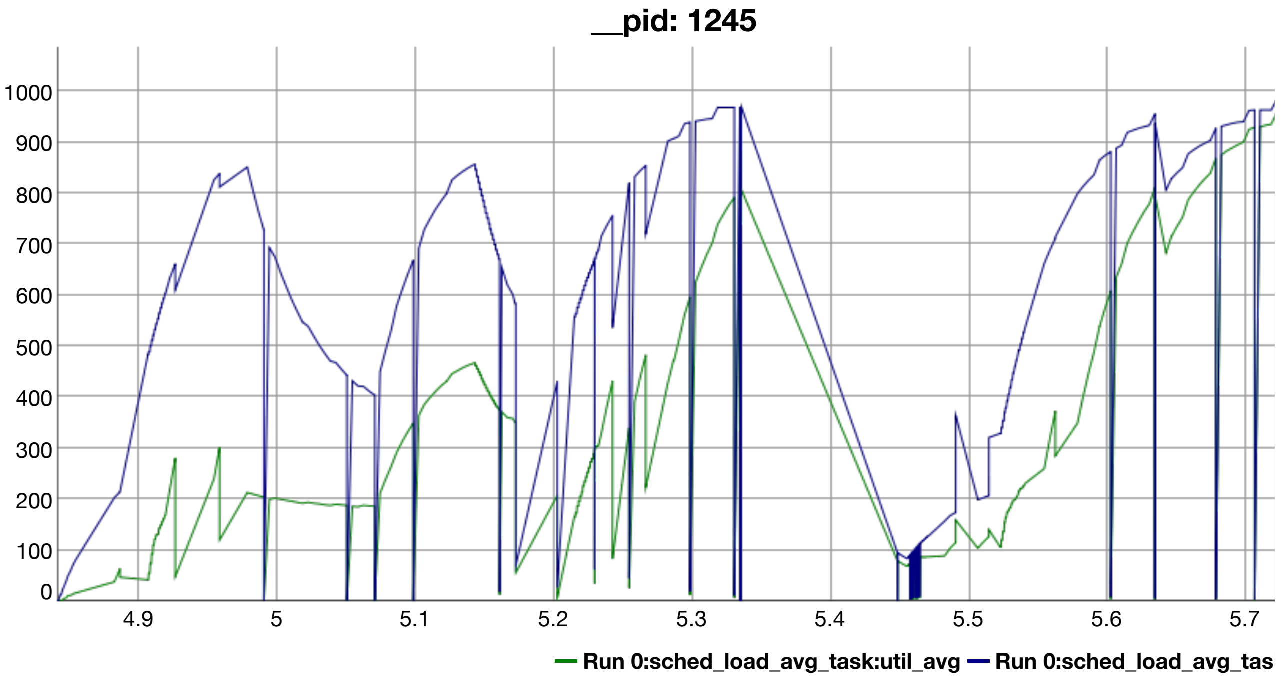


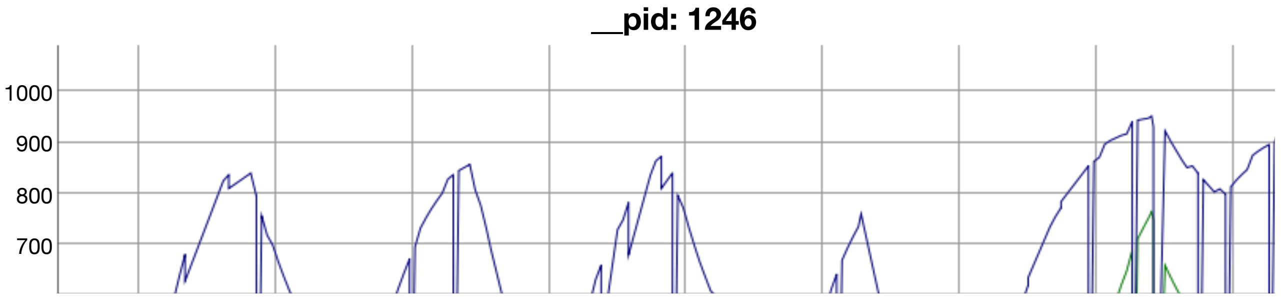
To see how PELT tracks CPU load and utilization, print RQ load and CPU utilization of the pid=1244 single-thread will help.

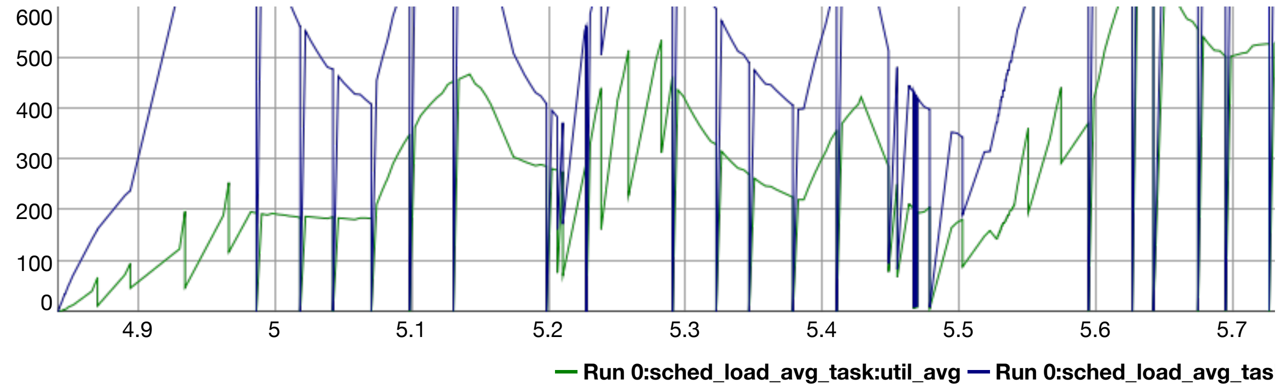
As expected, the load and utilizcation take about 32 ms to reach 512 (half of 1024). And it takes more the 200 ms for them to reach 1000.

In [7]:
```
l = trappy.ILinePlot(
            run,                                     # TRAPpy run Object
            trappy.sched.SchedLoadAvgTask,           # TRAPpy Event (maps to a unique word in
            column=[                                 # Column(s)
                "load_avg", "util_avg"],
            filters = {
                "__pid": [1244, 1245, 1246, 1247, 1248]
            },
            pivot="__pid",
            per_line=1)                              # Number of graphs per line
l.view()
```

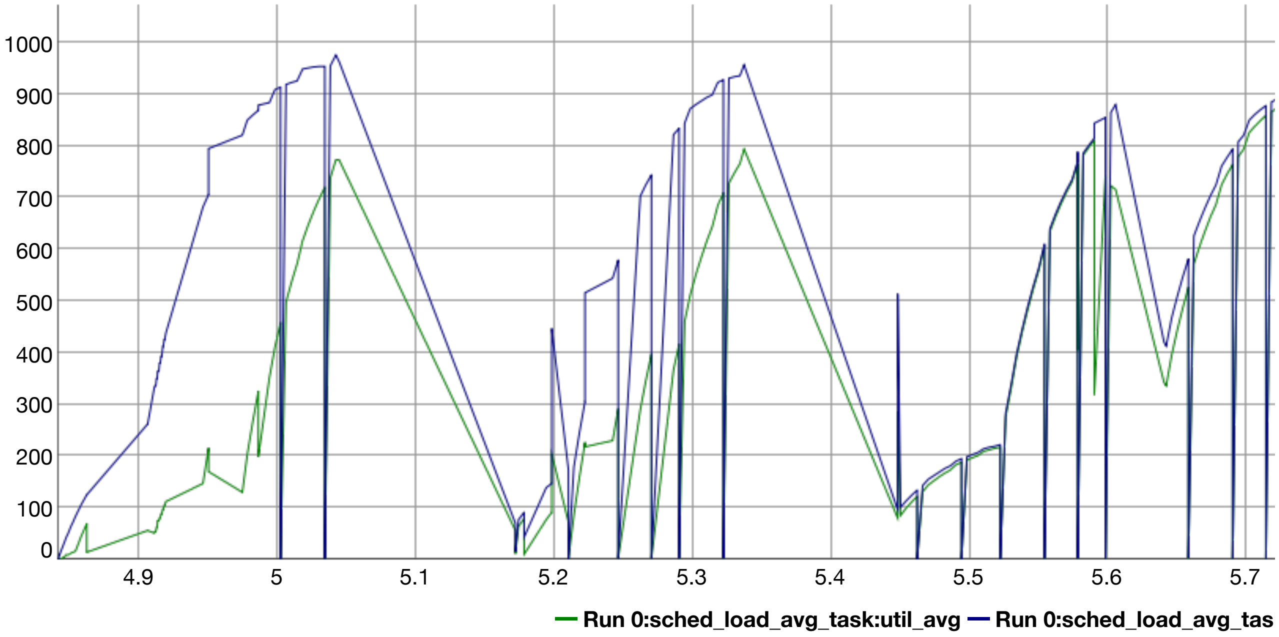




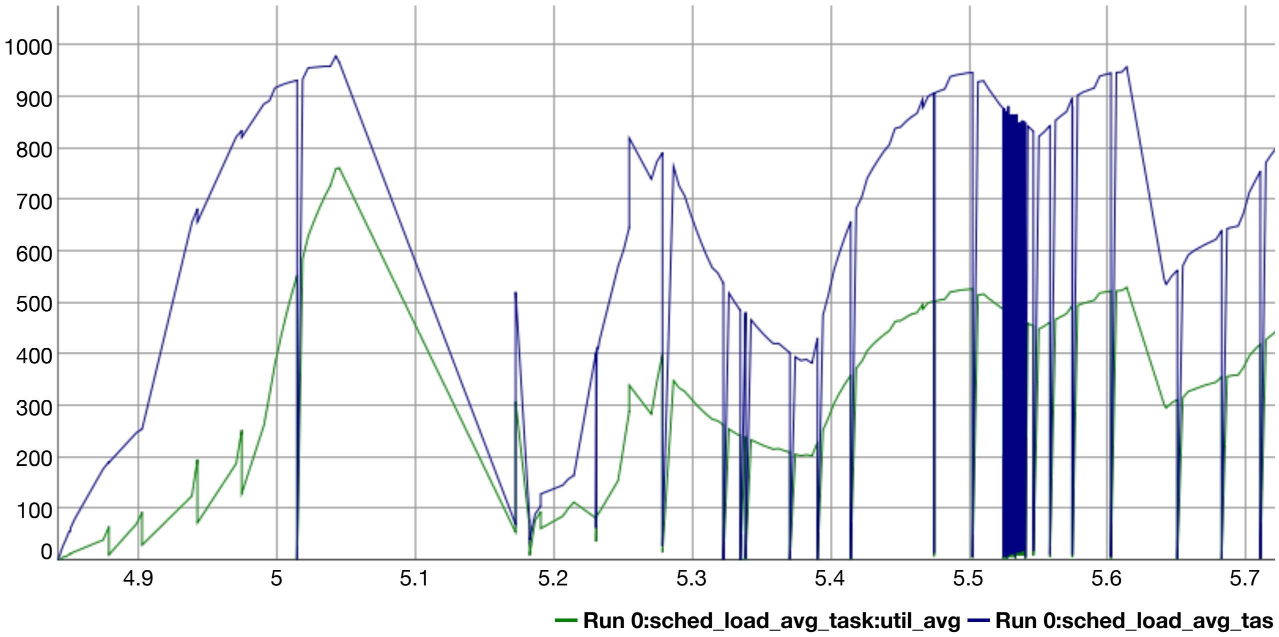




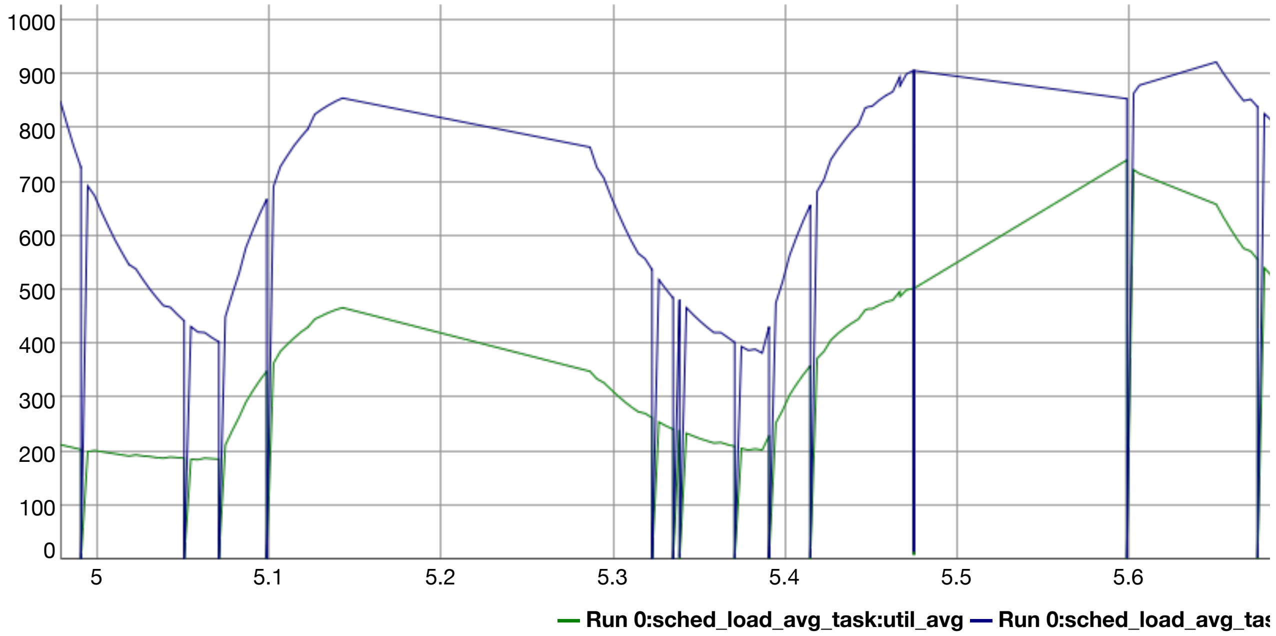
__pid: 1247



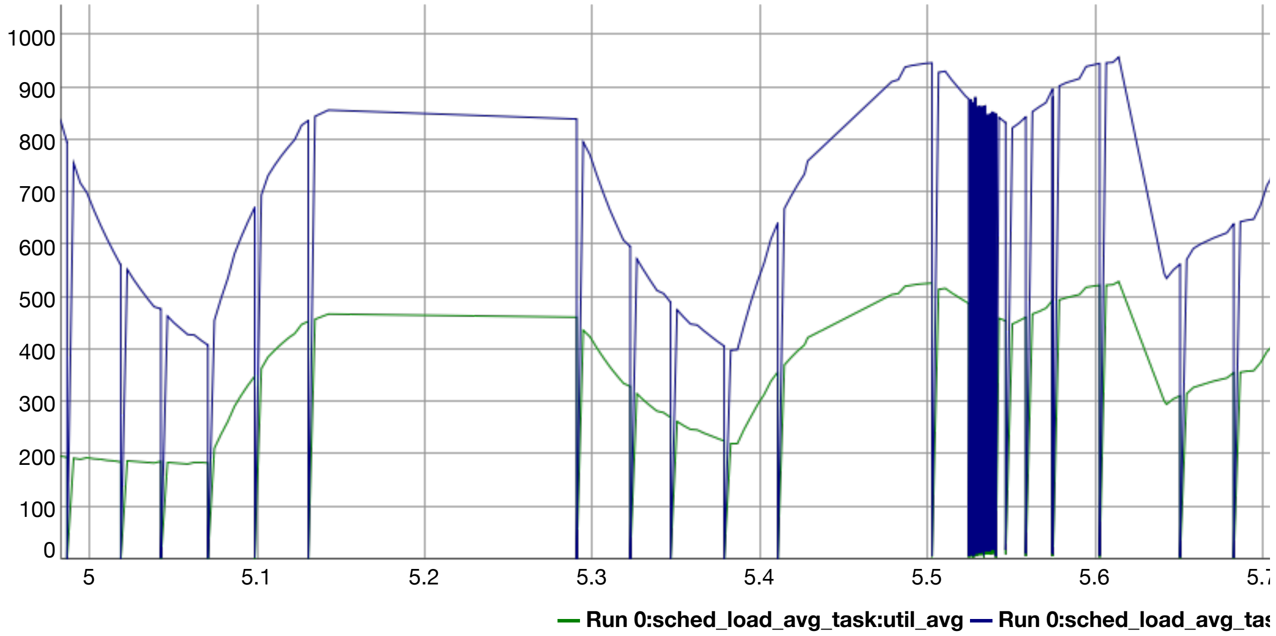
__pid: 1248



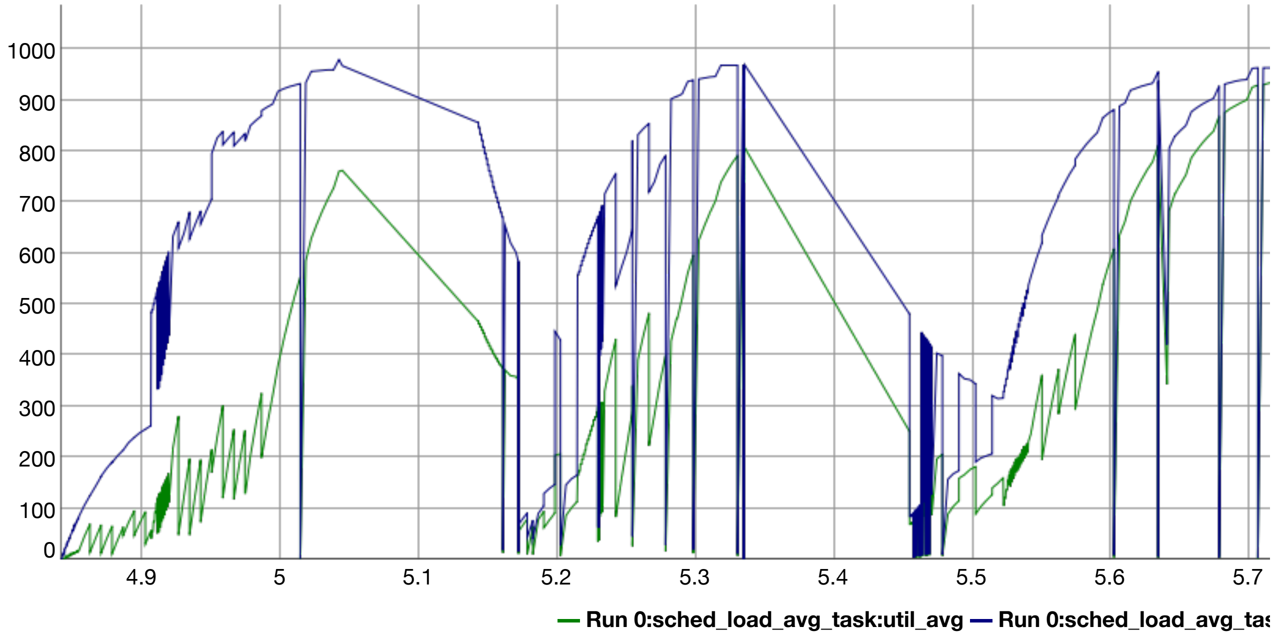
```
In [9]: l = trappy.ILinePlot(
                    run,                                        # TRAPpy run Object
                    trappy.sched.SchedLoadAvgTask,              # TRAPpy Event (maps to a unique word in
                    column=[                                    # Column(s)
                        "load_avg", "util_avg"],
                    filters = {
                        "__pid": [1245, 1246, 1247, 1248]
                    },
                    pivot="__cpu",
                    per_line=1)                                 # Number of graphs per line
        l.view()
```



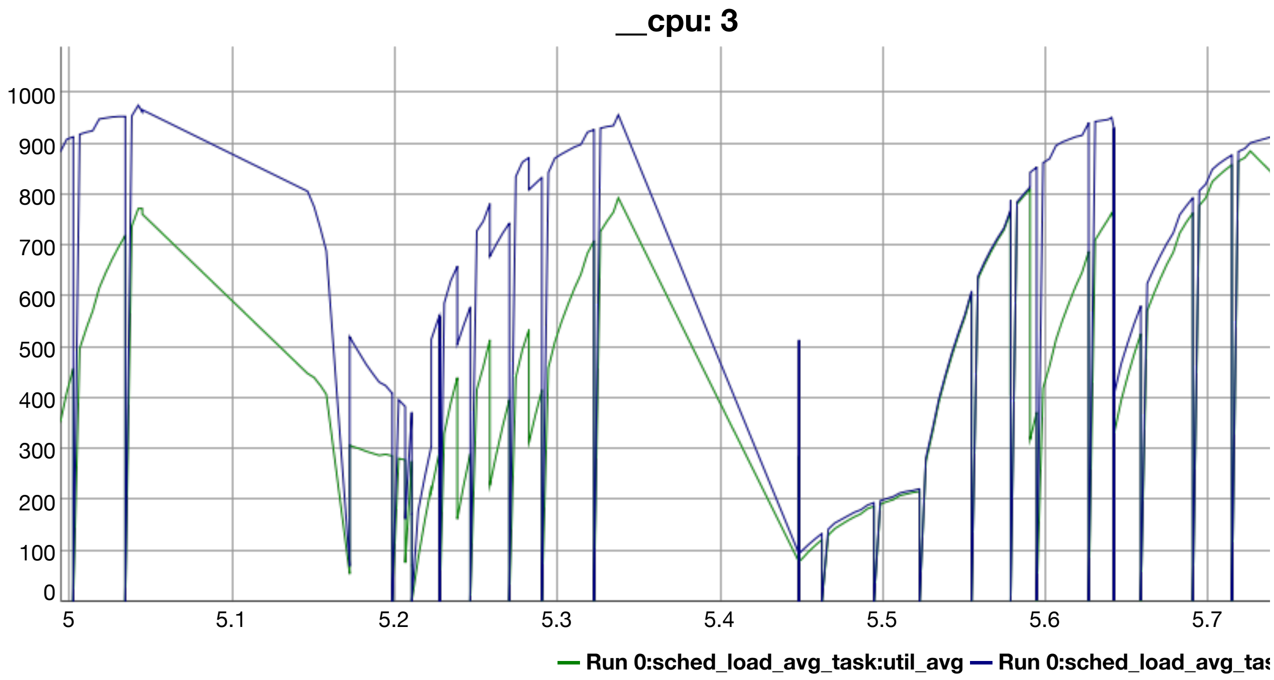
__cpu: 0



__cpu: 1



__cpu: 2


__cpu: 3
1000
900
800
700
600
500
400
300
200
100
0
5
5.1
5.2
5.3
5.4
5.5
5.6
5.7
Run 0:sched_load_avg_task:util_avg
Run 0:sched_load_avg_tas