Linaro

Improve utilization and load tracking in the scheduler

Presented by
Vincent Guittot

Date
November 3, 2016

Event
Linux Plumber Conference

Items

- Propagate tasks movement into task group
- Update idle CPUs
- Scale invariance
- Responsiveness of PELT

Propagate tasks movement into task group

Migrate inside task group

migration is not seen at root level

util_avg

Power Consumption or anything else you want to display

Current, Watt or whatever the unit

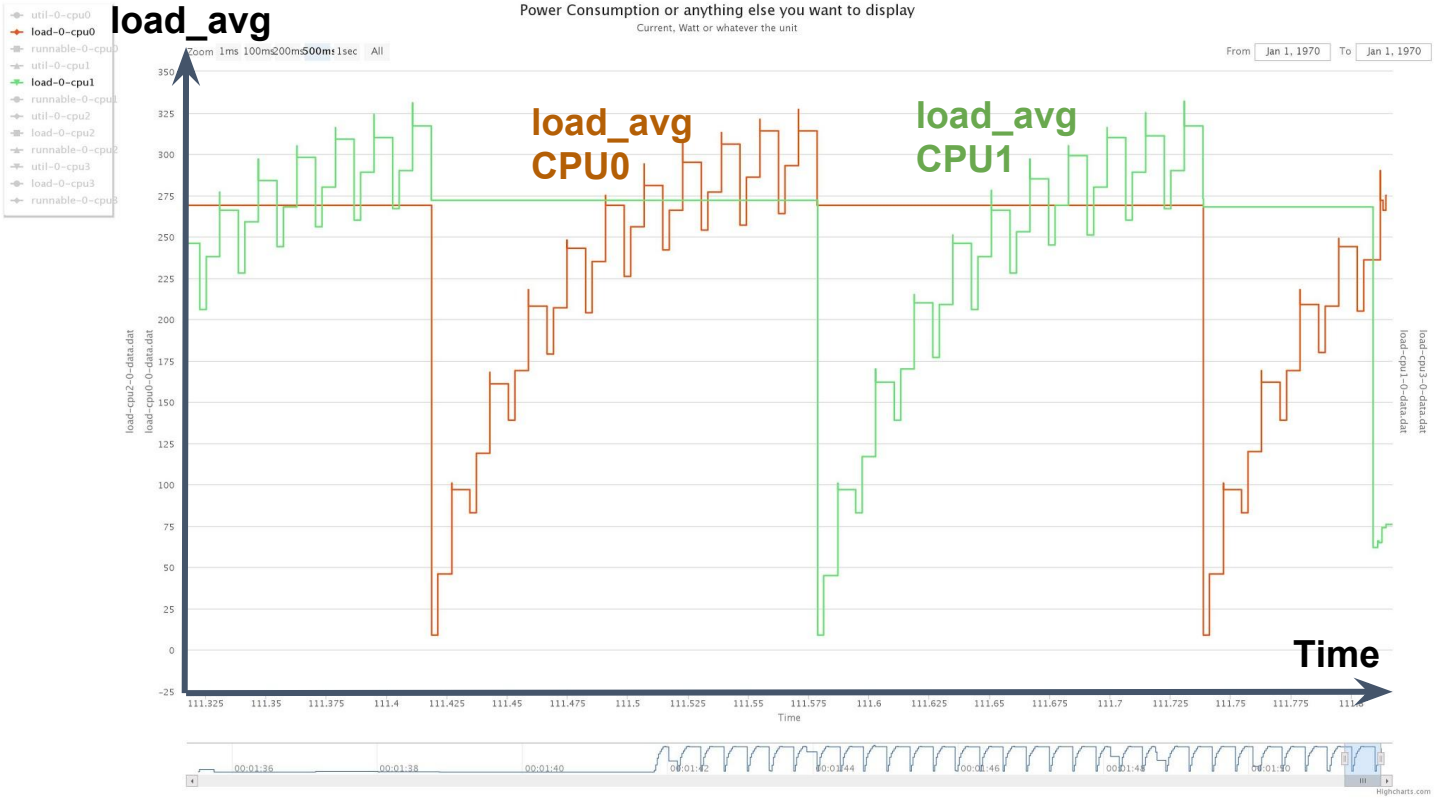
util_avg CPU0

util_avg CPU1

Time

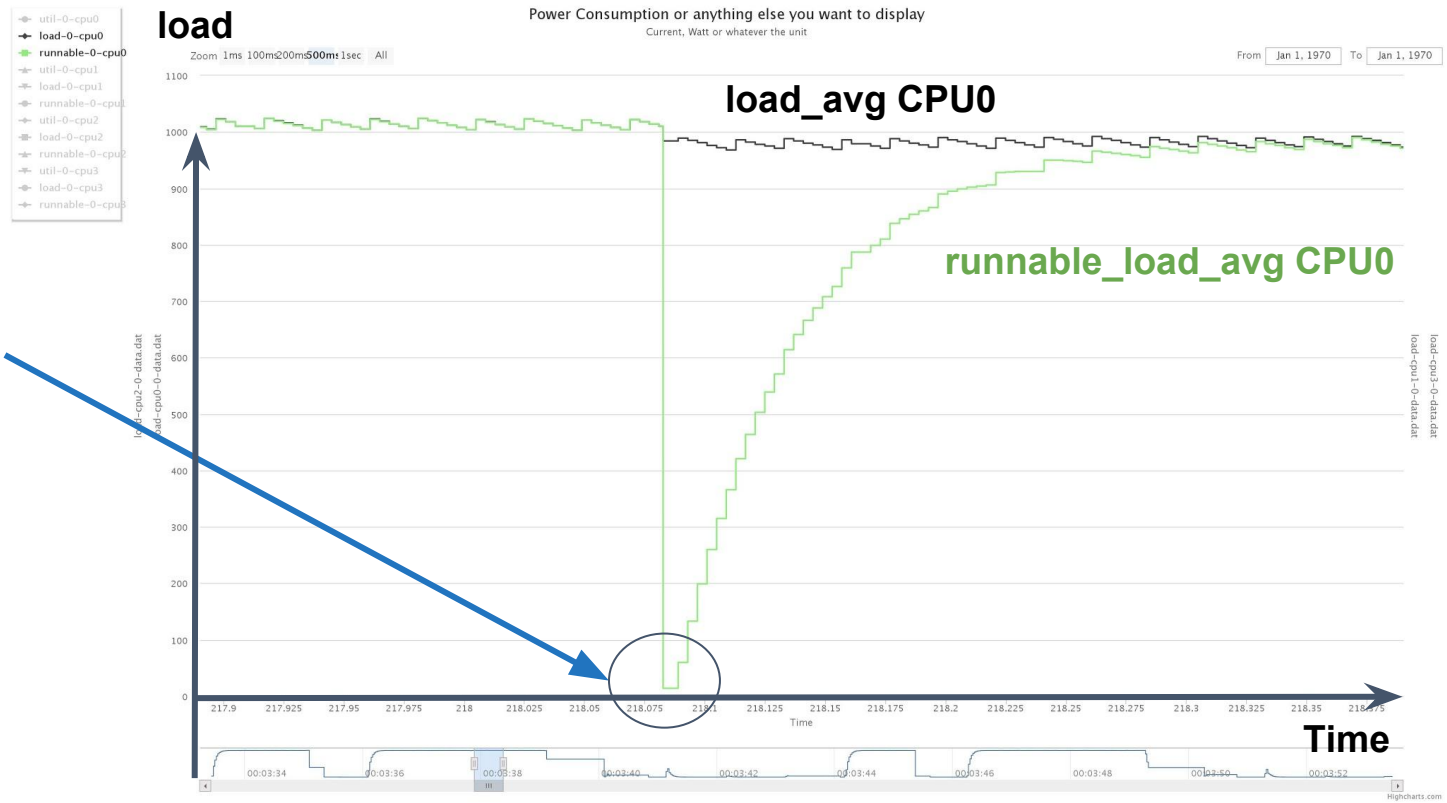
Migrate in task group

Same behavior for load



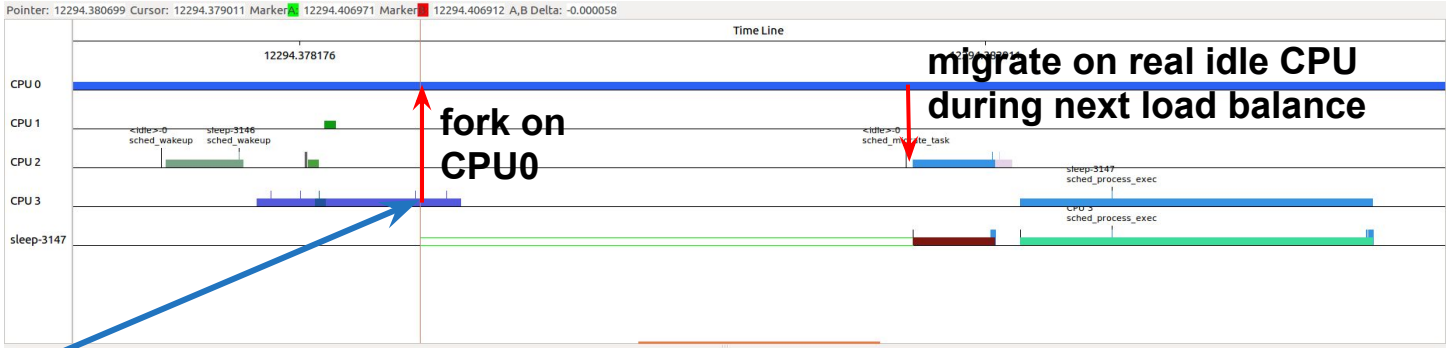
Move between task groups

runnable_load_avg becomes null



Move between task groups

loaded CPU can be selected as idlest CPU even when other CPUs are idle



Page Search: Column: # contains graph follows

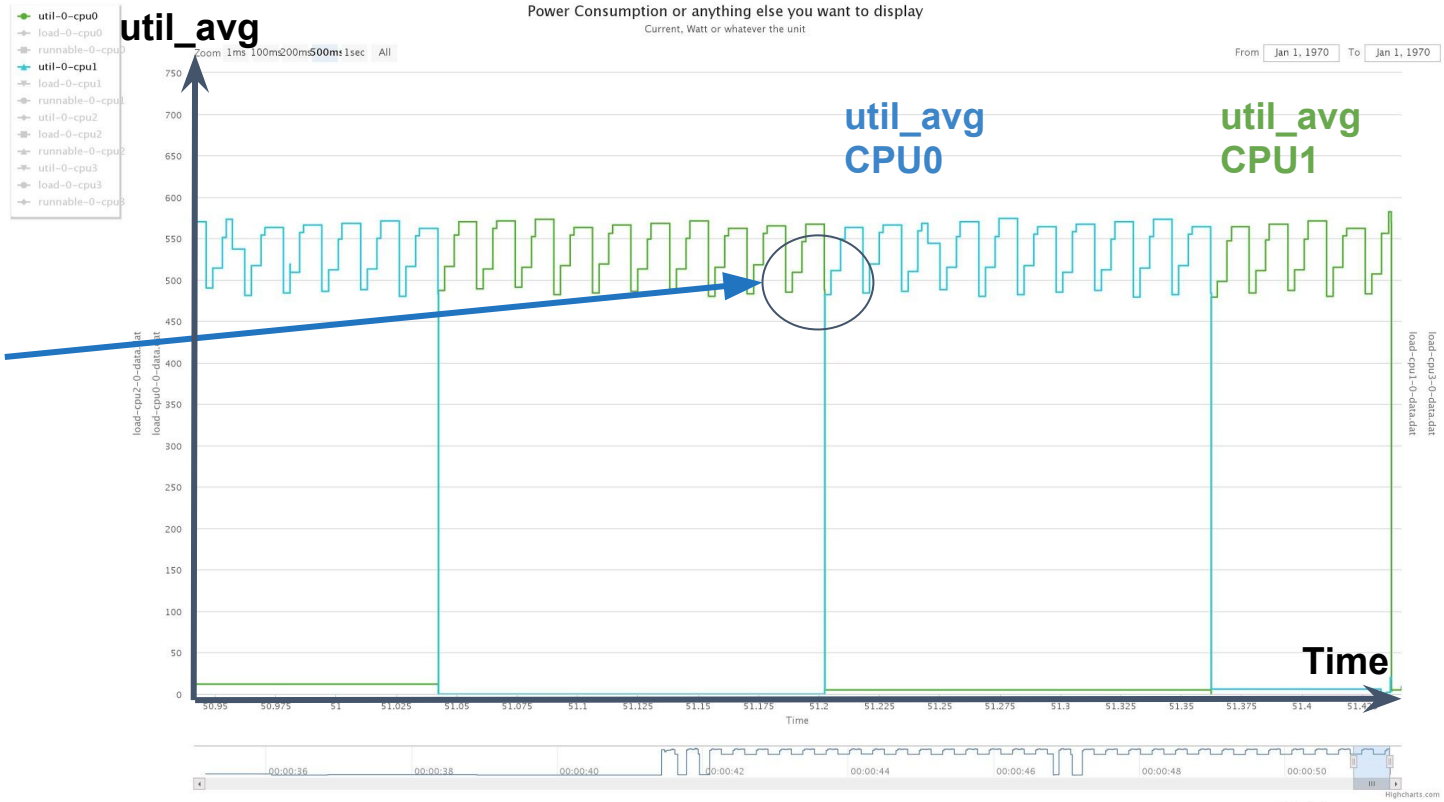
#	CPU	Time Stamp	Task	PID	Latency	Event	Info
172	2	12294.377760	sleep	3146	d..5	sched_wakeup	test-tg-6.sh:3129 [120] success=1 CPU:003
173	3	12294.377978	test-tg-6.sh	3129	....	sched_process_wait	comm=test-tg-6.sh pid=0 prio=120
174	3	12294.378184	test-tg-6.sh	3129	dNs3	sched_wakeup	kworker/3:2:1278 [120] success=1 CPU:003
175	2	12294.378214	<idle>	0	d.s3	sched_migrate_task	comm=rcu_preempt pid=7 prio=120 orig_cpu=3 dest_cpu=2
176	2	12294.378227	<idle>	0	dNs4	sched_wakeup	rcu_preempt:7 [120] success=1 CPU:002
177	3	12294.378315	kworker/3:2	1278	d..3	sched_wakeup	kworker/1:2:3060 [120] success=1 CPU:001
178	3	12294.378976	test-tg-6.sh	3129	....	sched_process_fork	comm=test-tg-6.sh pid=3129 child_comm=test-tg-6.sh child_pid=3147
179	3	12294.379011	test-tg-6.sh	3129	d..2	sched_wakeup_new	test-tg-6.sh:3147 [120] success=1 CPU:000
180	3	12294.379192	test-tg-6.sh	3129	....	sched_process_wait	comm=test-tg-6.sh pid=0 prio=120
181	2	12294.382366	<idle>	0	d.s2	sched_migrate_task	comm=test-tg-6.sh pid=3147 prio=120 orig_cpu=0 dest_cpu=2
182	2	12294.382963	sleep	3147	dN.3	sched_wakeup	migration/2:22 [0] success=1 CPU:002
183	2	12294.383012	migration/2	22	d..3	sched_migrate_task	comm=test-tg-6.sh pid=3147 prio=120 orig_cpu=2 dest_cpu=3
184	3	12294.383792	sleep	3147	....	sched_process_exec	filename=/bin/sleep pid=3147 old_pid=3147
185	2	12294.386330	<idle>	0	dNs3	sched_wakeup	rcu_preempt:7 [120] success=1 CPU:002
186	2	12294.394363	<idle>	0	dNs3	sched_wakeup	rcu_preempt:7 [120] success=1 CPU:002
187	2	12294.402257	<idle>	0	dNs3	sched_wakeup	rcu_preempt:7 [120] success=1 CPU:002
188	2	12294.410353	<idle>	0	dNs3	sched_wakeup	rcu_preempt:7 [120] success=1 CPU:002
189	2	12294.418250	<idle>	0	dNs3	sched_wakeup	rcu_preempt:7 [120] success=1 CPU:002
190	0	12294.842167	threadA	3137	dNs3	sched_wakeup	kworker/0:2:1301 [120] success=1 CPU:000
191	3	12295.385733	<idle>	0	dNh3	sched_wakeup	sleep:3147 [120] success=1 CPU:003
192	3	12295.386362	sleep	3147	d..4	sched_migrate_task	comm=test-tg-6.sh pid=3129 prio=120 orig_cpu=3 dest_cpu=1
193	3	12295.386385	sleep	3147	d..5	sched_wakeup	test-tg-6.sh:3129 [120] success=1 CPU:001

Propagate tasks movement into task group

- Propagate events down to the root domain
  - attach/detach task
  - remove task (asynchronous detach)
  - enqueue/dequeue task (already there)
- Propagate utilization
  - Simply add children changes to parent's utilization
- Propagate load
  - Task group must not consumes more CPU than a task of equal weight
  - Scale task group's load into the range of its shares

Propagate tasks movement into task group

Utilization migrates with task



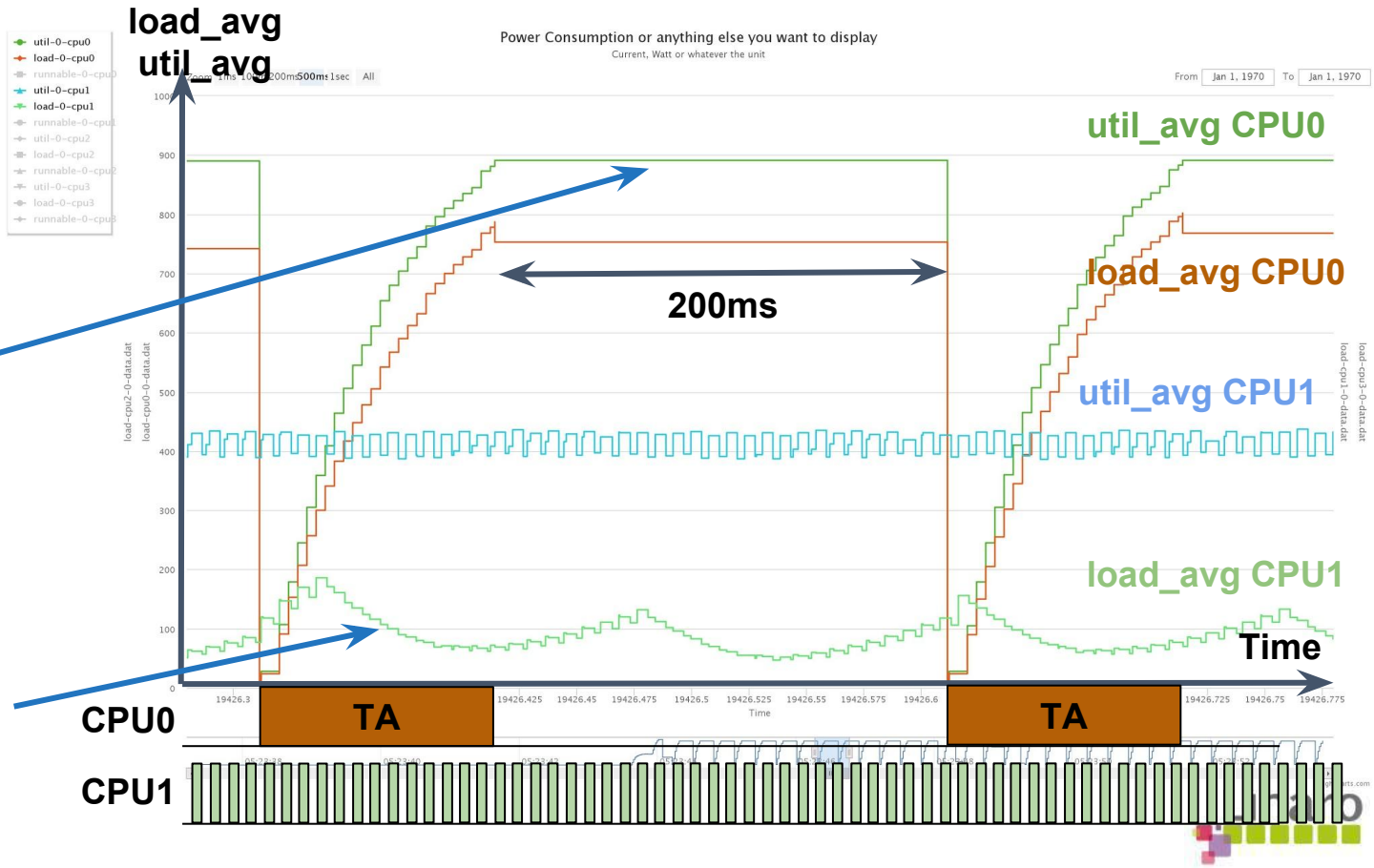
Linaro

Update idle CPUs

Blocked load/utilization of idle CPUs

load_avg/util_avg stay unchanged until next wake up or idle load balance

Impact the shares of busy CPUs

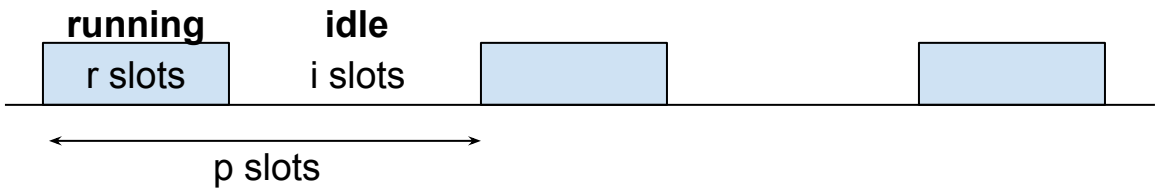


Update idle CPUs

- Ensure periodic update of blocked load
  - On an idle CPU like for periodic idle load balance

Scale invariance

Utilization and load of periodic task

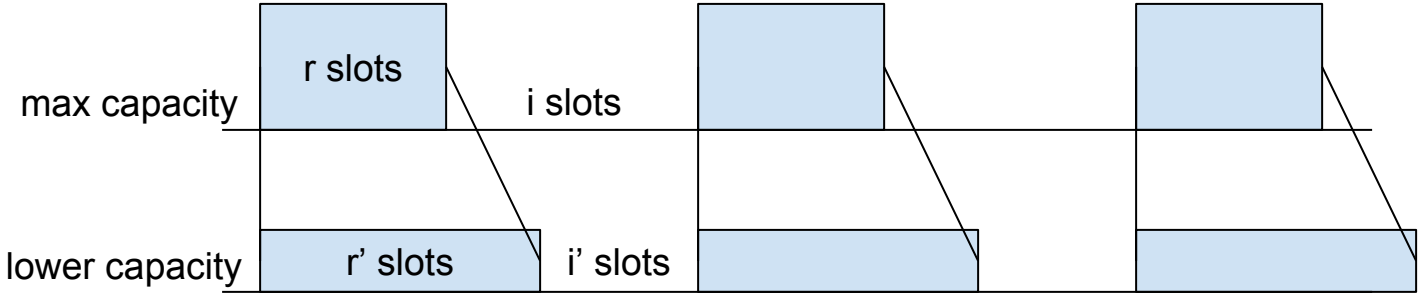


- Utilization range is defined by
  - Number of running slots: r
  - And number of idle slots: i

$$(1-y^r) / (1-y^p) * y^i < \text{Utilization} < (1-y^r) / (1-y^p)$$

- Similar for load

Scale utilization and load



- Invariance applies a scaling factor to the value of time slots
  - At max capacity the range is

$$(1-y^r) / (1-y^p) * y^i < \text{Utilization} < (1-y^r) / (1-y^p)$$

  - At lower capacity the range is

$$C * (1-y^{r'}) / (1-y^p) * y^{i'} < \text{Utilization} < C * (1-y^{r'}) / (1-y^p)$$

with C reflecting the compute capacity ratio

Scale utilization and load

Run 66ms at max capacity

Run 198ms at a 3rd of capacity

util_avg

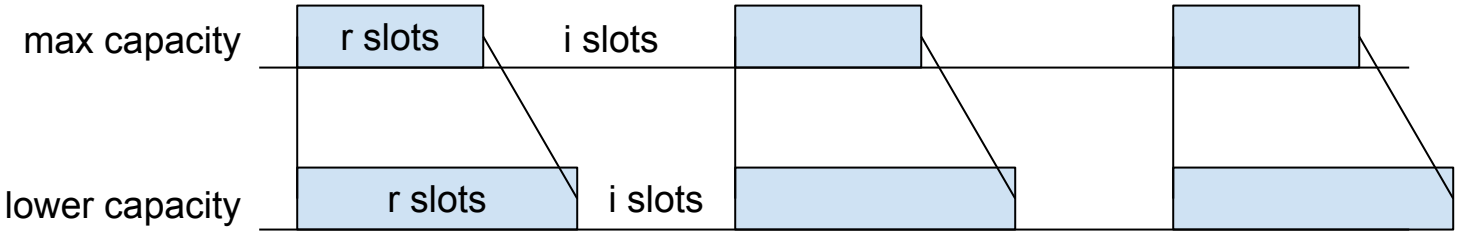
Power Consumption or anything else you want to display

Current, Watt or whatever the unit

Time

Scaled utilization for a task running 66ms every 330ms

Scale time



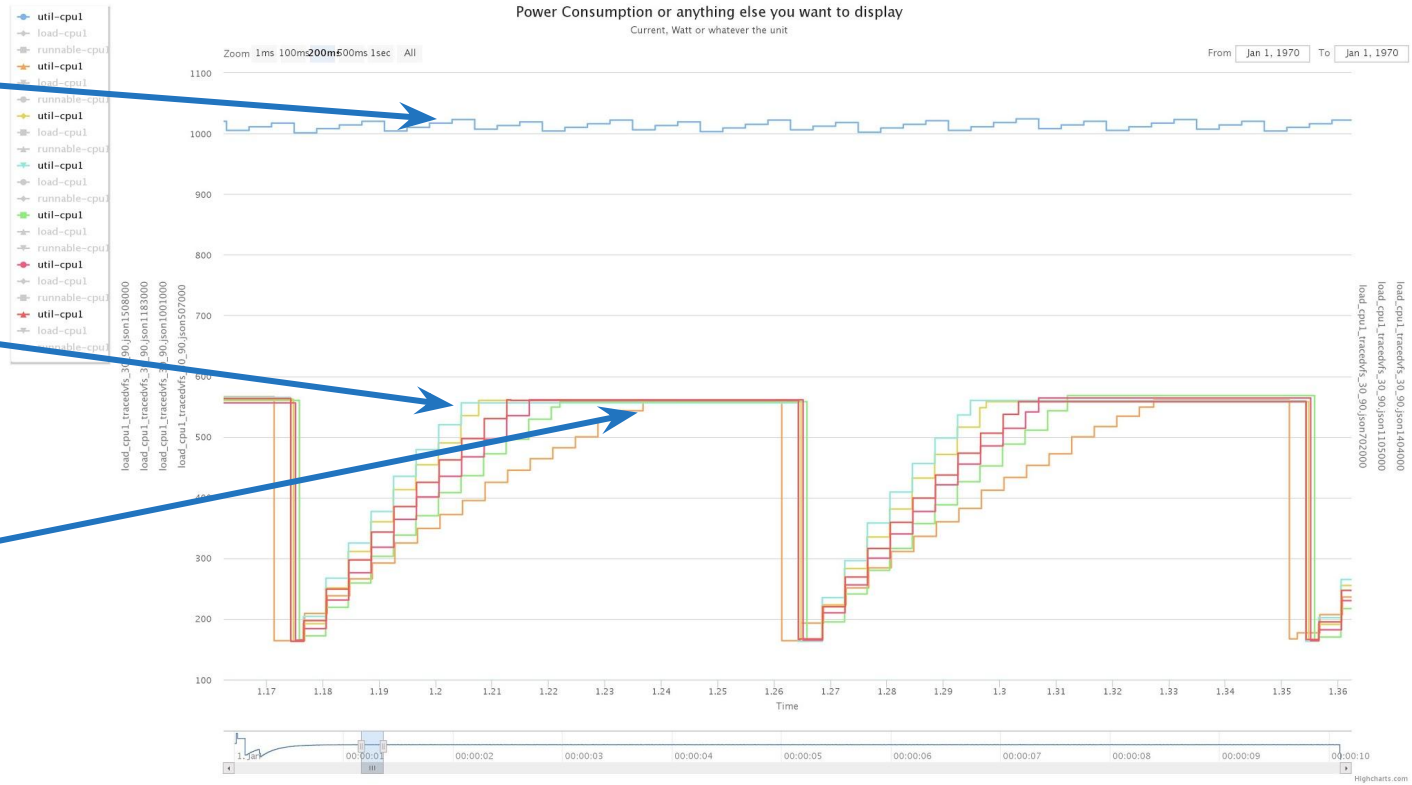
- Scale duration of time slot instead of scaling utilization of time slot

Scale time

Run always at a 3rd of capacity

Run 33ms at max capacity

Run 66ms at a half of capacity



Scaled utilization for a task running 33ms every 100ms

Linaro

Impact on load/utilization

- Utilization/Load no more capped by current capacity
  - they can reach max value but need more time
- Utilization/Load of a task is fully invariant
  - Same min/max value at wake up/sleep whatever the compute capacity

Capacity	Max	Half (mainline)	Half (new)	A third (mainline)	A third (new)
Util(%)					
972 (95%)	138ms	N/A	276ms	N/A	414ms
486 (47.5%)	30ms	138ms	60ms	N/A	90ms
256 (25%)	13ms	32ms	26ms	64ms	39ms

Responsiveness of PELT

Responsiveness of PELT

- Some figures
  - Time to reach 95%: 140ms
  - Time to reach 50% : 31ms

- Current decisions are proportional to utilization value
  - High Utilization value selects High OPP
  - Low Utilization value selects Low OPP

- Use other properties of utilization signal ?
  - Use variation
    - An increase of 100 in one running time means that it has run for at least 5ms
  - Apply some kind of PID algorithm
    - take into account variation of utilization

Other alternatives to improve responsiveness

- Use other scheduling class
  - deadline scheduler
- Run at fmax for some tasks
  - Boost utilization of some tasks and group of tasks
- Keep track of “peak per-entity utilization”
  - Track last max utilization value
- Use other load tracking mechanism