Agenda

- Task placement and load balancer tracepoints
- WALT tracepoints
- core_ctl/isolation tracepoints

Task placement & Load balancer trace points

sched_switch

sched_switch: prev_comm=gle.android.gms prev_pid=2769 prev_prio=120 prev_state=S ==> next_comm=Binder:1398_12 next_pid=2530 next_prio=120

prev_comm/prev_pid/prev_prio
- The name/pid/prio of the task that is getting scheduled out

prev_state
- **S** : normal/interruptible sleep
- **D** : disk/uninterruptible sleep
- **D|K** : killable uninterruptible sleep
- **R/R+** : preemption

next_comm/next_pid/next_prio
- The name/pid/prio of the task that is getting scheduled in

- prev_state = 'R' does not necessarily mean that a task is ***yielding***. When a task is executing in user mode and it gets preempted after servicing an IRQ, the state shows up as 'R', but it not yield. For all practical purposes, both 'R' and 'R+' means preemption.

- ***sched_switch*** trace point alone is sufficient to know how the loaded the CPUs are and which tasks are loading.

sched_wakeup

sched_wakeup: comm=.gms.persistent pid=2326 prio=120 target_cpu=001

comm/pid/prio
- The waking task name/pid/prio

target_cpu
- The CPU on which the task would be running

- ***sched_wakeup_new*** trace point is printed when a task is woken up for the first time after fork.
- The time duration between sched_wakeup tracepoint to sched_switch tracepoint in which this task apperas as next_pid is called ***scheduling latency***
- The time duration between ***sched_waking*** trace point to ***sched_wakeup*** trace point can be considered as the ***scheduler overhead***.

sched_enq_deq_task

sched_enq_deq_task: cpu=1 enqueue comm=kworker/u16:10 pid=910 prio=120 nr_running=3 cpu_load=79 rt_nr_running=0 affine=ff demand=2908473 pred_demand=2908473

cpu
- The CPU on which the task is enqueued/dequeued

enqueue/dequeue
- Is the task enqueued/dequeued

comm/pid/prio
- The name/pid/prio of the enqueued/dequeued task

nr_running
- The runnable tasks on this CPU after enqueue/dequeue

cpu_load
- The CFS runqueue runnable load avg (PELT)

rt_nr_running
- The runnable RT tasks on this CPU after enqueue/dequeue

affine
- The affinity of the task

demand/pred_demand
- Task's demand/predictive demand in nano sec

- This trace point is useful for checking the ***affinity*** settings of a task. For example, why a high demand task is running on the little cluster

- This trace point is added by us in SMP days. The ***cpu_load*** field should ***NOT*** be confused with any of the WALT signals.

sched_migrate_task

sched_migrate_task: comm=logd.writer pid=567 prio=130 load=26 orig_cpu=1 dest_cpu=2

comm/pid/prio
- The task name/pid/prio

load
- The task demand scaled to 1024

orig_cpu
- The task's previous CPU

dst_cpu
- The task's new CPU

- In HMP the load is printed in %. In EASz the load is scaled to 1024

- Enable this tracepoint if you want to findout the migration details on the fly. For example *grep 'sched_migrate_task.*orig_cpu=[0-3].*dst_cpu=[4-7]' | wc -l* gives the number of little to BIG micluster migrations.

- ***grep 'sched_migrate_task.*orig_cpu=[0-3].*dst_cpu=[4-7]' | wc -l*** gives the number of little to BIG micluster migrations.

sched_cpu_util

sched_cpu_util: comm=adsprpcd pid=974 cpu=0 task_util=63 nr_running=0 cpu_util=0 cpu_util_cum=71 new_cum_util=134 task_in_cum=0 capacity_curr=372 capacity=456 curr_util=23 sync=0 idle_state=0 irqload=3380935 high_irqload=0

---

comm/pid
- The waking task comm/pid

cpu
- The CPU being evaluated for this waking task

task_util
- The task demand (schedtune boost applied) scaled to 1024

nr_running
- The runnable tasks on this CPU

cpu_util
- The CPU's cumulative runnable average scaled to 1024

cpu_util_cum
- The CPU's cumulative window demand scaled to 1024

new_cum_util
- The CPU's cumulative window demand including the waking task

task_in_cum
- Did waking task run on this CPU in the current window

capacity_curr
- The current capacity of this CPU

capacity
- The capacity of this CPU. Does not affect task placement

curr_util
- The waker task demand (schedtune boost applied) scaled to 1024

sync
- Is it a sync wakeup

idle_state
- The c-state of this CPU

irqload
- irq/softirq load of this CPU

high_irqload
- irqload > sysctl_sched_cpu_high_irqload

sched_energy_diff_packing

sched_energy_diff_packing: comm=Binder:686_4 pid=1594 task_util=30 targeted_cpus=1 nrg_pack=0 nrg_spread=0 nrg_diff=0

comm/pid
- The waking task name/pid

task_util
- The task demand (schedtune boost applied) scaled to 1024

targeted_cpus
- The number of CPUs that can accommodate this task at the current capacity

nrg_pack
- The new capacity index if the task is packed

nrg_spread
- The new capacity index if the task is placed on an idle CPU

nrg_diff
- nrg_diff = nrg_pack - nrg_spread
- Task is placed on an idle CPU if nrg_diff != 0

- Enable sched_cpu_util, sched_energy_diff_packing, sched_task_util_* trace points to examine CPU selection algorithm. These trace points indicate why a CPU is selected for a given task.
- The CPU/cluster energy-model is available in procfs
- CPU0: /proc/sys/kernel/sched_domain/cpu0/domain0/group0/energy/cap_states contains the {capacity, frequency, energy} tuples of all available OPP.
- Cluster0: /proc/sys/kernel/sched_domain/cpu0/domain1/group0/energy/cap_states contains the {capacity, frequency, energy} tuples of all available OPP.

sched_task_util_energy_diff

sched_task_util_energy_diff: comm=Binder:2586_7 pid=3428 task_cpu=3 task_util=169 nominated_cpu=2 target_cpu=3 energy_diff=47 need_idle=0 latency=36302

comm/pid
- The waking task name/pid

task_cpu
- The task's previous CPU

task_util
- The task demand scaled to 1024

nominated_cpu
- which CPU is nominated by the wakeup selection algorithm

target_cpu
- Task is placed on which CPU? task_cpu/nominated_cpu

energy_diff
- The energy difference between task_cpu and nominated_cpu

need_idle
- is this task eligible for need_idle (PF_WAKEUP_IDLE)

latency
- The time taken in usec to run the wakeup selection algorithm

- ***sched_task_util_energy_aware***: When the previous CPU is not eligible (isolation, reserved, hosting need_idle tasks, high irqload), energy_diff() is skipped and nominated CPU is selected as target
- ***sched_task_util_bias_to_waker***: The sync wakee is placed on the waker CPU. energy_diff() is skipped.
- ***sched_task_util_colocated***: The co-location enabled tasks are placed on the nominated CPU without running energy_diff()
- ***sched_task_util_boosted***: The placement eligible tasks are placed on the nominated CPU (least loaded CPU in the BIG cluster) without running energy_diff()
- ***sched_task_util_overutilzed***: The task is placed on the nominated CPU if the previous CPU can't accommodate this task under overutilization

sched_group_energy

sched_group_energy: cpu=0 group_util=89 total_nrg=31744 busy_nrg=2759 idle_nrg=28985 grp_idle_idx=0 new_capacity=93

cpu
- The first CPU in the group for which energy is being evaluated

group_util
- The sum of the cumulative window demand of all CPUs in the group

total_nrg
- The estimated total energy (busy + idle) for this group

busy_nrg
- Busy energy is estimated by factoring the energy @ new capacity into the group utilization

idle_nrg
- Idle energy is estimated by factoring the energy @ group idle index into the (1024 - group utilization)

grp_idle_idx
- The idle index of the shallowest CPU in the group

new_capacity
- The capacity of this group considering if the task is placed/migrated

- Enable this tracepoint to understand why the nominated CPU is rejected. It generates lots of traffic.

- This trace point is printed twice for each group traversed during energy evaluation. one for when task is placed on the previous CPU and another for when the task is placed on the nominated CPU

- For example, when the previous CPU and nominated CPU are both in the same cluster, we iterate 5 groups. 4 groups for 4 CPUs and 1 group for cluster. So this trace point is printed 10 times

sched_load_balance

sched_load_balance: cpu=1 state=idle balance=0 group=0x0 busy_nr=0 imbalance=0 flags=0x0 ld_moved=0 bal_int=16

cpu
- The CPU that is doing load balance (destination_cpu)

state
- busy/newly_idle/idle

group
- The cpumask of busy/src CPUs
- Either a CPU or cluster in our systems

busy_nr
- The runnable tasks on the busy/src CPU after this event

imbalance
- The load difference between src and dest after this event

flags
- LBF_ALL_PINNED, LBF_SOME_PINNED, LBF_NEED_BREAK
- LBF_MOVED_RELATED_THREAD_GROUP_TASK
- defined in kernel/sched/fair.c

```
#define LBF_ALL_PINNED  0x01
#define LBF_NEED_BREAK  0x02
#define LBF_SOME_PINNED 0x08
#define LBF_MOVED_RELATED_THREAD_GROUP_TASK 0x400
```

ld_moved
- How many tasks are migrated as part of this lb

bal_int
- When is next balance? In msec

sched_overutilized

sched_overutilized: overutilized=1
sched_overutilized: overutilized=0

---

overutilized
- Is one or more CPUs overutilized in the system?

- Energy aware task placement is ***NOT*** disabled under overutilization condition
- The load balancer restrictions are turned ***OFF*** under overutilization condition
- The no-HZ idle balancer kicks are allowed only when a CPU is overutilized

WALT trace points

sched_get_task_cpu_cycles

sched_get_task_cpu_cycles: cpu=3 event=5 cycles=50388000000 exec_time=198854 freq=253391 legacy_freq=1593600

cpu
- The CPU for which the update is called

event

```
enum task_event {
        PUT_PREV_TASK   = 0,
        PICK_NEXT_TASK  = 1,
        TASK_WAKE       = 2,
        TASK_MIGRATE    = 3,
        TASK_UPDATE     = 4,
        IRQ_UPDATE      = 5,
};
```

cycles
- cycles = actual_cycles_for_this_update * NSEC_PER_MSEC

exec_time
- The time in nano seconds accounted for this update

freq
- cycles/freq kHZ

legacy_freq
- The current frequency as per cpufreq

- This trace point is really helpful to know the current operating frequency under ***thermal***. The cpufreq does not know about the throttling frequency
- The computed frequency does ***NOT*** give the current frequency, but it is an average frequency. It closely follows the current frequency for TASK_UPDATE or PUT_PREV_TASK events
- The computed frequency has higher ***error*** when measured in short intervals

sched_update_task_ravg_mini

sched_update_task_ravg_mini: wc 310930168629 ws 310920000001 delta 10168628 event TASK_UPDATE cpu 3 task 0 (swapper/3) ms 310930080140 delta 88489 demand 0 rq_cs 487179 rq_ps 1835941 cur_window 0 prev_window 0 grp_cs 0 grp_ps 0

wc
- The current time at the update of this event

ws
- The window start of the CPU

delta
- delta = wc - ws

event
- PUT_PREV_TASK/PICK_NEXT_TASK/TASK_WAKE
- TASK_MIGRATE/TASK_UPDATE/IRQ_UPDATE

cpu
- The CPU for which the update is called

task
- The task for which the update is called

ms
- The task's mark_start

delta
- delta = wc - ms

demand
- The tasks demand in nano sec

rq_cs
- The CPU's busy time in the current window

rq_ps
- The CPU's busy time in the previous window

cur_window
- The task's busy time in the current window

prev_window
- The task's busy time in the previous window

grp_cs
- The CPU's group busy time in the current window

grp_ps
- The CPU's group busy time in the previous window

sched_load_to_gov

sched_load_to_gov: cpu=1 policy=0 ed_task_pid=-1 aggr_grp_load=2829551 freq_aggr_thresh=200000000 tt_load=2840000 rq_ps=822649 grp_rq_ps=2829551 nt_ps=0 grp_nt_ps=0 pl=3907360 load=3652200

---

cpu
- The CPU for which the load is reported

policy
- Frequency reporting policy
- top-task/cpu_load/*max(top-task/cpu_load)*

ed_task_pid
- The task (if any) pid that activated early detection

aggr_grp_load
- The aggregated group (schedtune.top-app) load in this cluster

freq_aggr_thresh
- The frequency aggregation threshold
- 1000% by default. 1% when sched_boost = 3

tt_load
- The load/utilization of top-task

rq_ps
- This CPU's prev_runnable_sum

grp_rq_ps
- This CPU's group prev_runnable_sum

nt_ps
- This CPU's new task prev_runnable_sum

grp_nt_ps
- This CPU's new task group prev_runnable_sum

pl
- The predictive load of this CPU

load
- The reported load of this CPU

```
if (ed_task_pid)
        load = sched_ravg_window; /* 100% */
if (aggr_grp_load > freq_aggr_thresh)
        load = rq_ps + aggr_grp_load;
else
        load = rq_ps + grp_rq_ps;
load = max(load, tt_load);
```

sched_set_boost

165819.207548: sched_set_boost: type 1
165821.226730: sched_set_boost: type 0
165821.226919: sched_set_boost: type 3
165834.248407: sched_set_boost: type 0

---

type

- sched_boost type

```
#define NO_BOOST                0
#define FULL_THROTTLE_BOOST     1
#define CONSERVATIVE_BOOST      2
#define RESTRAINED_BOOST        3
```

sched_set_preferred_cluster

sched_set_preferred_cluster: group_id 1 total_demand 2075646 preferred_cluster_first_cpu 0

---

group_id
- The id of the co-location group id
- The schedtune.top-app group id will always be 1

total_demand
- The sum of co-location demand of all tasks in this group

preferred_cluster_first_cpu
- The first CPU of the preferred cluster for this group

core_ctl/isolation trace points

cpu_isolate

sched_isolate: iso cpu=5 cpus=0x30 time=2430 us isolated=1
sched_isolate: iso cpu=5 cpus=0x10 time=1476 us isolated=0

---

iso cpu
- The CPU that is being isolated/un-isolated

cpus
- cpu_isolated_mask after this operation

time
- time (in usec) taken for this operation

isolated
- isolation/un-isolation

core_ctl_set_boost

4746.248763: core_ctl_set_boost: refcount=1, ret=0
4748.735025: core_ctl_set_boost: refcount=0, ret=0

---

refcount
- boost is active when refcount > 0
- boost is inactive when refcount = 0

ret
- return 0 upon success
- return -EINVAL when refcount is mismatched

core_ctl_set_busy

core_ctl_set_busy: cpu=4, busy=82, old_is_busy=0, new_is_busy=1

busy
- CPU utilizaiton (busy time) in % for the last window
- scaled to the original capacity (@ Fmax) of the CPU

old_is_busy
- is this CPU busy previously?

new_is_busy
- is this CPU busy now?

```
if (busy >= busy_up_thres || high_irqload)
        new_is_busy = true;
else if (busy < busy_down_thres)
        new_is_busy = false;
else
        new_is_busy = old_is_busy
```

- The CPU busy stats are updated for all CPUs when the window is rolled over
- If a CPU has high irqload, it is considered as busy irrespective of its utilization

sched_get_nr_running_avg

sched_get_nr_running_avg: avg=2 big_avg=0 iowait_avg=0 max_nr=2 big_max_nr=0

---

avg
- The running average of the runnable tasks over the last window

big_avg
- The running average of the runnable BIG tasks over the last window

iowait_avg
- The running average of the iowait tasks over the last window

max_nr
- The instantaneous maximum number of runnable tasks on any CPU in the last window

big_max_nr
- The instantaneous maximum number of runnable tasks on any BIG CPU in the last window

core_ctl_eval_need

core_ctl_eval_need: cpu=4, old_need=2, new_need=3, updated=1
core_ctl_eval_need: cpu=4, old_need=3, new_need=2, updated=0

old_need
- The number of CPUs needed back when the last adjustment is done

new_need
- The number of CPUs needed at the moment

updated
- is core_ctl/X thread woken up to do the adjustment?

```
updated = false;
if (new_need > active_cpus)
        updated = true;
else if (new_need == old_need )
        need_ts = curr_time;
else if (need_ts - curr_time > offline_delay)
        updated = true;
```

- Enable ***cpu_isolate, sched_get_nr_running_avg, core_ctl_set_busy, core_ctl_eval_need*** trace points to know why core_ctl bringing too many/less CPUs out of isolation

THE END