ZONE, EAS 1.1 AND EAS-Z

LINUX SCHEDULER TEAM
linux.sched.core@quicinc.com
Presented by Vikram Mulukutla

WHERE'S MY PPT?

- A note on this slide deck - it was created with reveal.js which the author(s) feel(s) is far more flexible and powerful than Powerpoint
- Zone details are in red , EAS details are in blue, and EAS-Z details are in turquoise
- Navigation - Use arrow keys; each topic may have 'vertical' slides. Press ESC to see the full layout
- Contribution - You can correct/contribute! Just do a regular pull request at [link]

INTRODUCTION - ZONE

- The HMP/Zone scheduler has been successfully commercialized in several QC SoC solutions, providing us with the ability to meet power/perf goals on varied CPU and CPUSS architectures. Easily the most prolific on ARM-licensed mobile SoCs with b.L
- Includes a custom load tracking solution and modifications to core task placement, frequency guidance and load balancer behavior, plus competitive value-adds
- The only real caveats have been divergence from android-common and upstream, and a per-SoC tuning effort from all parties

INTRODUCTION - EAS

- Upstream Linux doesn't - inspite of running on billions of such devices - support b.L or other heterogenous mobile architectures. ARM, OEMs and vendors (us too) need to do a better job upstreaming, plain and simple
- EAS is ARM's solution. An effort that started out in ~2012, with Pixel (8996) being the first commercial product to adopt it (EAS 1.1)
- In spite of extremely slow progress and upstream adoption, EAS is still the best hope toward a fully upstream solution
- EAS 1.2 is out, it's not a major upgrade - we are evaluating it currently

INTRODUCTION - WHY ARE WE SWITCHING

- Reducing the cost of software is of paramount importance. Convergence of fundamental codebases such as the Linux scheduler is one way to achieve this

- A common foundational scheduler allows Google and our OEM customers to vastly reduce per-vendor tuning and will prefer EAS going forward

- Open-source is give and take. Build a strong community with external scheduler experts without losing competitive advantage.

- Enter EAS-Z. We take the lessons learnt from HMP/Zone, apply them on top of EAS-Z, with our competitive value adds. Porting and evaluation with power/perf/APT took about 3 months

FEATURE COMPARISON

Feature	HMP/Zone	EAS	EAS-Z
Load Tracking	WALT	PELT / WALT	WALT
Task Placement CPU stats	Instantaneous (WALT cr-avg)	Long-term (PELT) instantaneous (WALT)	Instantaneous (WALT cuml_demand, cr-avg)
Load Balancing	Aggressive	Minimal (overutil)	In-between
Top Task	Yes	No	Yes
Predictive DCVS	Yes	No	Yes
Colocation, Freq Aggr.	Yes	No	Yes
Schedtune	No	Yes	Yes
Sched Boost	Yes	No	Yes
CPU Isolation, Core Control	Yes	No	Yes
IRQ Load Detection	Yes	No	Yes
Early Detection	Yes	No	Yes

WHERE IS THE CODE?

Branch	Zone	EAS1.1	EAS1.2	EAS-Z
msm-4.4	Enabled	Disabled	N/A	N/A
msm-4.9	Disabled	Enabled with EAS-Z	N/A	Enabled
android-3.18	N/A	Enabled	N/A	N/A
android-4.4	N/A	N/A	Enabled	N/A *
android-4.9	N/A	Enabled	N/A	N/A

* we are working on getting some of our non-competitive features into android-4.4 and then into Wahoo.
It's not going to be EAS-Z :-)

OVERUTILIZATION

- Is a new concept in EAS. A CPU is overutilized if its utilization is greater than 80% of it's capacity. If one CPU is overutilized, the entire system is considered overutilized
- 'Utilization' is meant to be actual execution time of tasks on CPU. The PELT signal is called util_avg
- With WALT and EAS, utilization for task placement and freq guidance is cpu_load
- With WALT and EAS-Z, utilization for task placement equals cumulative cpu demand [cpu_util()], and utilization is cpu_load for frequency guidance [cpu_util_freq]. Also overutilization threshold is 95%

CAPACITY MARGINS

- EAS uses a fixed capacity margin of 80% everywhere. This is primarily used when checking if a task fits on a CPU.

- EAS-Z (currently) uses a capacity_margin of 95% (so when upmigrating this is the threshold), and a capacity_margin_down of 85% (downmigrating)

- With respect to cpu capacity margin EAS-Z (currently) uses a capacity_margin_freq of 80% to match the governor (target_load)

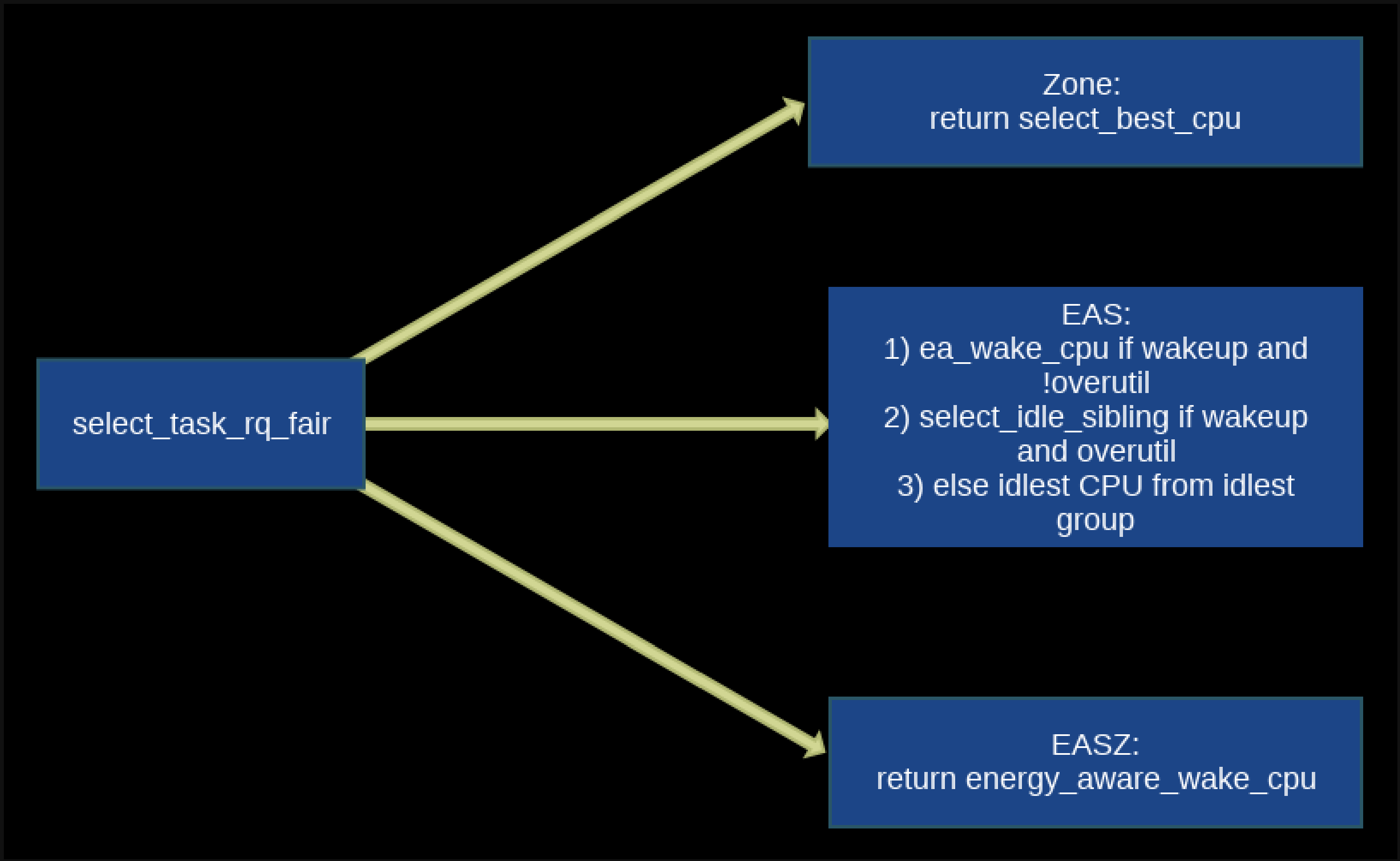
SCHEDTUNE

- EAS uses a single tunable called schedtune. This is basically an admission that ultimately, the scheduler can't tell if a thread is actually important
- .... the scheduler needs userspace to tell it which groups of tasks are important
- Task utilization is scaled to 1024, and if you have a schedtune % of 25%, boosted task utilization works out this way:
  task_util_boosted = task_util + 25% * (1024 - task_util)
- Boosted cpu utilization is decided by the highest-boosted-task on the runqueue
- We will evaulate schedtune with EAS-Z. It's currently designed to not require schedtune just like Zone, but obviously there are usecases that may benefit

CFS WAKEUP PLACEMENT: EAS

- EAS relies on the concept of an 'energy model' that is fairly accurate for a given SoC. Uses the energy model to estimate the total energy change of a system when placing a task on a CPU or moving one between CPUs
- Wakeup energy_aware placement is intended to minimize energy cost while not sacrificing latency
- Will only occur when the system is not overutilized
- Energy-aware placement will not be done for newly forked tasks or on exec

CFS WAKEUP PLACEMENT: COMPARISON



CFS WAKEUP PLACEMENT: EAS1.1 SKELETON

```
if (sync_hint_enabled)
    return current-CPU

if big.L:
    target_cpu = perform_big_little_cpu_selection
else
    target_cpu = perform_8996_cpu_selection

if (schedtune_boosted or prefer_idle and target_cpu is idle)
    return target_cpu

if target_cpu not found:
    return prev_cpu

else if target_cpu != prev_cpu:
    if (overutilized(prev_cpu))
        return target_cpu
    if (energy_diff(target_cpu) >= 0)
        return prev_cpu
return target_cpu
```

CFS WAKEUP PLACEMENT: EAS1.1 SKELETON

```
perform_big_little_cpu_selection()
   target_cpu = -1

   target_cluster = find cluster that can fit the task

   target_cpu = find cpu in target_cluster that can fit the task.
                for cpus that fit task at curr_capacity:
                    select first non-idle CPU
                    else select any idle CPU
                else fallback to CPU that fits task at higher OPP
   return target_cpu
```

CFS WAKEUP PLACEMENT: EAS1.1 SKELETON

```
perform_8996_cpu_selection() /* find_best_target */
   for_all_cpus (N-1 to 0 if schedtune_boosted, else 0 to N-1):

       Skip if task can't fit at max_freq
       Skip if high irq-load
       if prefer_idle, remember idle cpu as best_idle_cpu
           continue if cpu was idle
       if task can fit on cpu:
           if cpu is not idle:
               target_cpu = select least util cpu if prefer_idle
               else target_cpu = select highest util cpu
           else if prefer_idle:
               select cpu with shallowest c-state
               and set as best_idle_cpu
       else
           find backup cpu with least capacity

   return one of
       best_idle_cpu, target_cpu or backup cpu
       -1 if none available
```

CFS WAKEUP PLACEMENT: EAS1.1 WALT STATS

- When doing wakeup placement you need to know task utilization as well as cpu utilization to compare the two
- EAS uses WALT task demand as task utilization
- EAS uses prev_runnable_sum as cpu utilization, which is the right idea for frequency selection but quite wrong when doing task placement.
- This is because task demand includes wait-time and is an average value, whereas prev_runnable_sum doesn't include wait-time and is the previous window's stat.

CFS WAKEUP PLACEMENT: EAS-Z

- EAS-Z also relies on the concept of an 'energy model', we're not changing this central concept
- EAS-Z sacrifices some wakeup-selection overhead in order to get things right immediately
- Energy aware placement will *always* be done in select_task_rq_fair. This is so that we can respect colocation settings and always use c-state awareness etc.

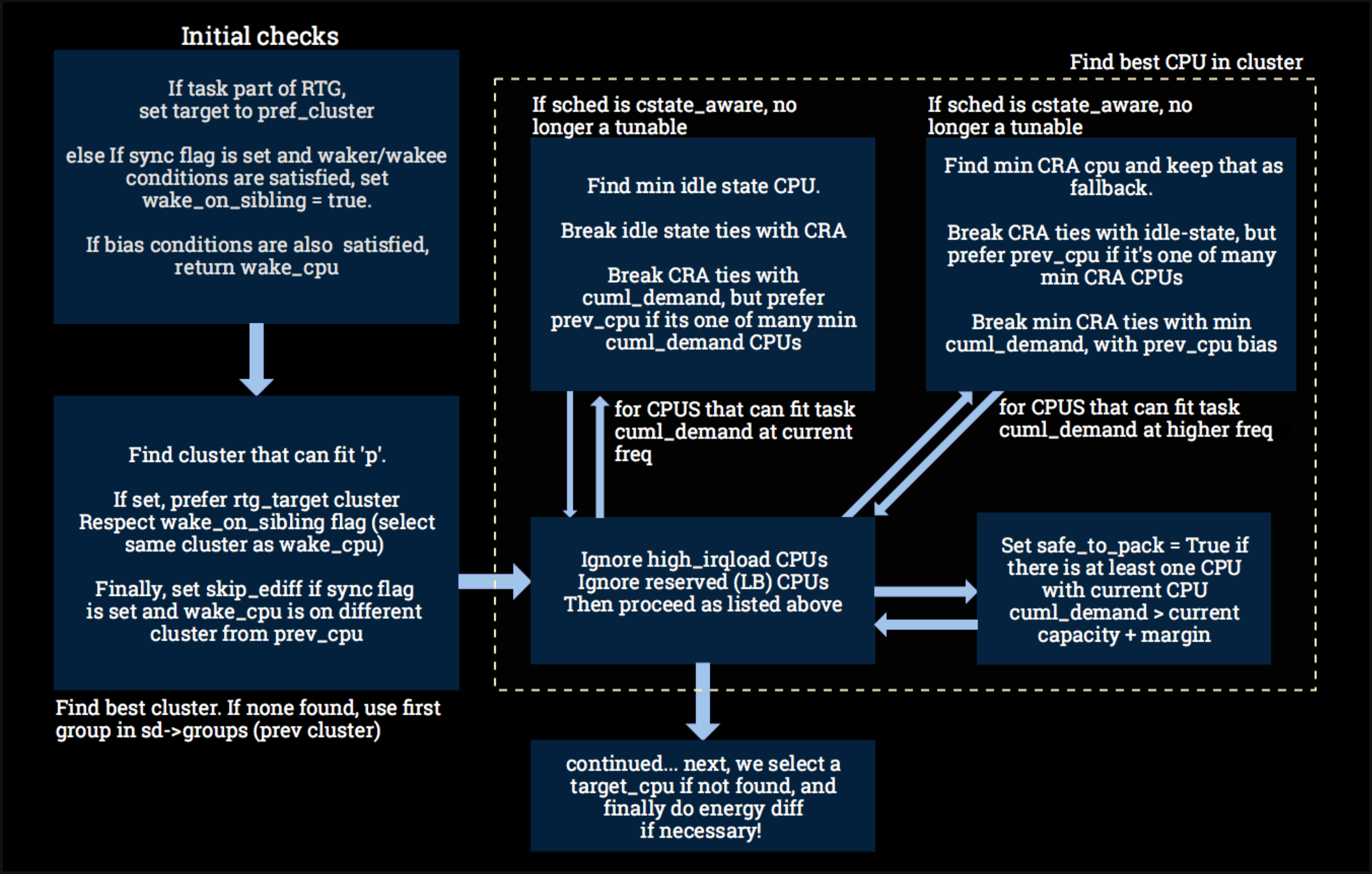
EAS-Z WAKEUP PLACEMENT: CUMULATIVE_WINDOW_DEMAND

- *Uses a new WALT CPU load metric called cumulative_window_demand, which is the sum of task demands in a window, even those that are off the runqueue.*

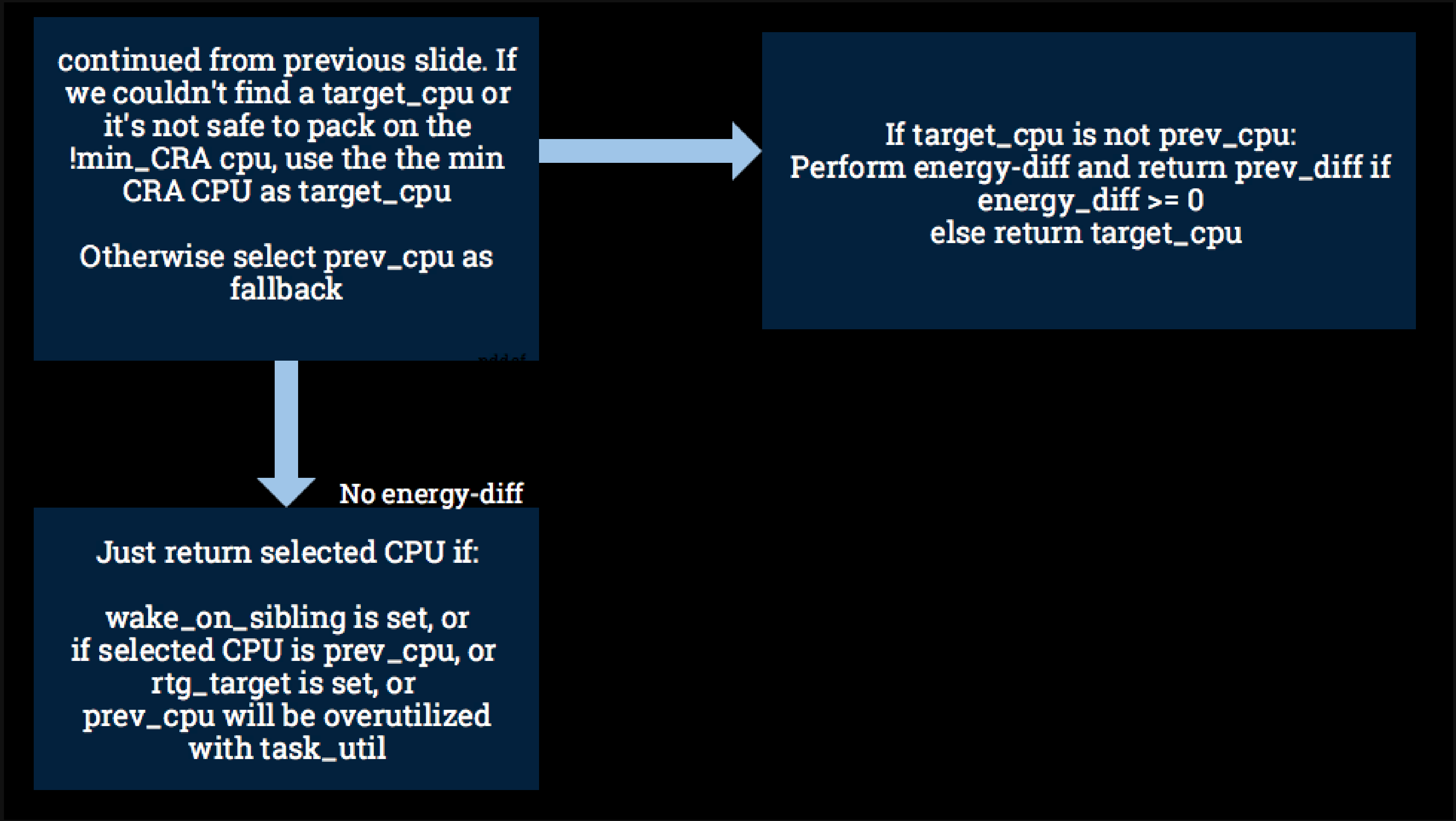
- Less instantaneous than CRA, since it considers tasks that slept very recently

- Remember that CRA (cumulative_runnable_avg) is the sum of task demands of tasks currently on the runqueue. Cumulative_window_demand is simply set to CRA on window rollover

CFS WAKEUP PLACEMENT: EAS-Z



CFS WAKEUP PLACEMENT: EAS-Z



ENERGY MODEL

- {capacity, energy} tuples for CPUs & clusters at each OPP. Energy data comes from SPT.

```
max_capacity = 1024 * cpu_fmax/max_possi_freq * cpu_ipc/max_possi_ipc
capacity_at_freq = max_capacity * freq/cpu_fmax
```

- And energy cost of keeping a CPU/cluster in each idle {c,d}-state (NOT cost of entering/exiting c-state)

- Example:

```
CPU_COST_0:
    busy-cost-data =
         92   34      /*  300000 */
         .
        490  152      /* 1593600 */
    idle-cost-data =
        22 18 14 12
CLUSTER_COST_0:
    busy-cost-data =
         92   3       /*  300000 */
         .
        490  15       /* 1593600 */
    idle-cost-data =
        4 3 2 1
```

ENERGY DIFF

- Here's the basic algorithm for calculating the cost of moving or adding a task to a CPU. sched_groups are arranged such that you will encounter one group with just one cpu, or one group with all CPUs in the cluster

```
energy_diff():
        for_each_sched_group:
                if (dest_cpu or source_cpu in sched_group):
                        energy_before +=
                            group_energy(before adding/moving util)
                        energy_after +=
                            group_energy(after adding/moving util)
        energy_diff = energy_after - energy_before
```

- Here's the basic algorithm for group_energy

```
group_energy():
        new_group_util = util after moving task to/away-from CPU
        new_capacity = capacity after freq change
        group_busy_energy = new_group_util * energy(new_capacity)
        group_idle_energy = (1024 - new_group_util) * energy(idle_state)
        return group_busy_energy + group_idle_energy
```

ENERGY DIFF - EASZ

- EAS-Z modifies group_energy to account for the fact that idle state 0 doesn't mean a c-state, and should be counted as 'busy' energy

```
group_energy():
        // ngu is new_group_util
        ngu = util after moving task to/away-from CPU
        new_capacity = capacity after freq change
        group_busy_energy = ngu * energy(new_capacity)
        group_idle_energy = if idle_state is 0:
                                ngu * energy(new_capacity)
                            else:
                                (1024 - ngu) * energy(idle_state)
        return group_busy_energy + group_idle_energy
```

LOAD BALANCING

LB type	Zone	EAS	EAS-Z
Idle Load Balance	Enabled with restrictions	Disabled until overutilized	Disabled until overutilized
Newly Idle Load Balance	Enabled with restrictions	Disabled until overutilized	Enabled with restrictions
Active Load Balance	regular load balance path and uses sbc() in tick()	Disabled until overutilized and then via regular load balance	regular load balance path and uses ea_wake_cpu() in tick()
Busy Balance	Enabled with restrictions	Disabled until overutilized	Disabled until overutilized

LOAD BALANCING - FIND THE BUSIEST GROUP

- find_busiest_group() is invoked by the scheduler to find the busiest group during LB ops
- Zone updates a group's total load (sum of cpu_load) and keeps track of BIG tasks. These are used to skip intercluster LB if necessary, or force active-balance of BIG tasks.
- EAS: mark runqueues as "overutilized" in this path. EAS **stops LB** if system is not overutilized! If overutilized, force-balance busiest group has a misfit task
- EAS-Z: allow LB to continue if we're doing a NEWLY_IDLE load balance with no overutil. Similar to Zone, we also use capacity stats, cumulative_win_demand to skip intra/intercluster balancing as necessary

LOAD BALANCING - FIND THE BUSIEST RQ

- find_busiest_queue(fbq) is invoked by the scheduler to find the busiest runqueue during load balance operations. Tasks are pulled from this runqueue
- Zone's fbq_hmp() logic finds a cpu with the greatest CRA. If performing idle or newly-idle LB, it prioritizes CPUs that have BIG tasks (active balanced later)
- EAS relaxes the existing SMP constraints on in fbq() for load balancing runqueues that have a single task on them, if that task is 'misfit'
- EAS-Z (in addition to EAS above) also tries to prioritize misfit task CPUs rather than CPUs with many smaller tasks, akin to the 'BIG' task concept

ACTIVE LOAD BALANCE - NEED_ACTIVE_BALANCE()

- Zone forces active-balance if there are BIG tasks that need balancing (flag set earlier in the find_busiest_group() path)

- EAS checks that rt-scaled and max capacity of source cpu is lower than that of dest cpu, and that dest cpu is not overutilized, and there is one CFS task that has caused the source cpu to be overutilized. If all true, it performs active LB.

- EAS-Z checks that max capacity of source cpu is less than that of destination CPU and source cpu contains at least one 'misfit' task, i.e. it's more relaxed than EAS, and along similar lines to the Zone 'BIG task' concept

ACTIVE LOAD BALANCE - CHECK_FOR_MIGRATION()

- check_for_migration() is a QC addon that (re)uses the wakeup path CPU selection, invoked via the tick() and schedboost paths
- Zone has addl. checks including capacity, irq-load, boost, nice values...
  - Finally uses sbc to figure out target.
- EAS-Z checks if the rq has a misfit task flag set, checks for single-cpu affinity and then calls energy_aware_wake_cpu to select target cpu...
  - .. and rejects lower capacity CPUs.
- Both EAS-Z and HMP mark the target-cpu as reserved to disallow wakeup placement on that CPU until the task has actually been pushed onto it.

SHOULD WE WAKE UP A CPU TO DO LB (NOHZ_KICK_NEEDED)?

- nohz_kick_needed() is used to check if we need to wake up an idle cpu to perform load balancing on behalf of the current CPU. Getting it wrong equals a sizable power hit

- HMP only kicks an idle cpu awake if
  - number if nr_running tasks >= 2, and spill restrictions are satisifed OR sched_boost is enabled

- EAS kicks an idle CPU if RQ has more than 2 tasks, OR there's >= 1 CFS task and there's RT (non-CFS) pressure on the CPU

- EAS-Z will only kick if rq->nr_nunning >= 2 AND CPU is overutilized, OR there's >= 1 CFS task and there's RT (non-CFS) pressure on the CPU (more conservative)

WHICH IDLE CPU DO WE WAKE UP?

- find_new_ilb() is used to find an idle CPU to perform load balancing on behalf of the 'calling' CPU
- Zone attempts to pick an idle CPU 'closest' to the CPU we're performing idle LB on behalf of. If spill restrictions were found to be enabled earlier, Zone checks that it isn't kicking a higher power-cost CPU
- EAS just uses pure-SMP here and selects the lowest numbered idle CPU in nohz.idle_cpus_mask (which can include cpus from all clusters)
- EAS-Z attempts to first use the lowest numbered idle CPU in the same cluster as the 'calling' CPU. If that doesn't work, we fall back to the EAS selection IFF the calling CPU is overutilized or the calling CPU is a BIG CPU

CAN_MIGRATE_TASK CHECKS

- So after all the work of figuring out imbalance, finding busiest group, rq, etc., you might *still* not want to LB a task away from its cpu...
- can_migrate_task is the last line of defense for a task against LB
- Zone tries to avoid migrating
  1. preferred_cluster tasks,
  2. BIG Tasks that don't fit on the destination CPU, and
  3. small tasks from lower capacity cpus when those CPUs have big tasks on them
  4. BIG tasks from 'busiest' group if the group's total capacity can support those number of tasks

- EAS-Z does (1) and (2), but not (3) and (4)
- When !overutilized, EAS-Z also attempts prevents NEWLY_IDLE migration of a task if it would cause cumulative demand on the destination to increase

FREQUENCY GUIDANCE

- As noted earlier, HMP and EAS-Z both use WALT exclusively, and both report one or more of the following stats to the governor depending on policy:
  - cpu_load + group_cpu_load
  - (with freq aggregation) cpu_load + aggregated_group_load
  - Max (Top Task Load, cpu_load + group_cpu_load)
  - Top Task Load
  - New Workload
  - Early Detection Load

- Predictive DCVS Load is supported by EAS-Z as well

FREQUENCY GUIDANCE - LOAD REPORTING DETAILS

- With zone plus interactive, load is reported via a timer that runs every window_size nanoseconds. Load is reported for both inter and intra-cluster migrations, if expected frequency change >= 400MHz.

- EAS-Z has been tested with schedutil. There is no timer - instead, EAS-Z uses an irq-work scheduled from the tick path to report load. The window rolls over just after a tick; this allows the latest load to be reported to the governor asap.

- EAS-Z does not notify the governor on intracluster migrations, and always notifies the governor on intercluster migrations

FREQUENCY GUIDANCE - GOVERNOR DETAILS

- Schedutil is meant to use scheduler load stats directly when deciding a perf level. No 'sampling' involved; is instead driven by sched events such as tick(), migrations etc. See this wiki for more on load scale conversions
- DCVS and sched teams worked together to make schedutil use WALT stats
- Interactive has various tunables and quirks such as hispeed, above_hispeed_delay, target_load, max_freq_hysterisis etc.
- Schedutil supports hispeed alone for now

FREQUENCY GUIDANCE - GOVERNOR DETAILS

- next_freq = cpu_max_freq * 1.25 * util/max
  // 20% headroom
- See go/schedtogov for more details
- util may be a combination or one of the stats in the slides above
- Work is done in SCHED_FIFO context

RT PLACEMENT: EAS

- EAS RT placement does nothing special in the RT placement path
- ...which is slightly odd considering the {re}surge{nce} of interest in RT tasks
- So let's take a look at SMP RT placement. A bitmask called the 'lowest' mask is constructed which contains CPUs that have tasks with <= priority to the RT task being placed.
- The previous CPU is prioritized, then a CPU that is logically closest in the cache topology, skipping smp_processor_id if it isn't in the set of CPUs that are considered 'lowest'

RT PLACEMENT: ZONE VS EAS

- Zone RT placement attempts to find a CPU with low IRQ load that also has the minimum cumulative_runnable_avg.
- It also respects spill settings.

- Furthermore, if the RT task is a short-burst task, Zone will attempt to find a CPU with the least wakeup latency.

- Finally, ties due to identical load are broken with considerations to previous cpu or a CPU that shares a cache with the previous CPU.

- Enabling FULL_THROTTLE_BOOST with SCHED_BOOST_ON_BIG will cause Zone to place RT tasks on the big cluster

RT PLACEMENT

- EAS-Z RT placement finds the max capacity cluster if FULL_THROTTLE_BOOST with SCHED_BOOST_ON_BIG is enabled, otherwise finds the least capacity cluster
- EAS-Z RT placement avoids high irqload CPUs, and overutilized CPUs
- Attempts to find a CPU in the least idle state that can fit the Rt task with current capacity. Ties are broken with cumulative demand on the CPU
- As a backup, we use the CPU with the maximum spare capacity as fallback. If this fails as well, then we go back to SMP-style CPU selection

VALUE ADDS

Value Add	Zone	EAS-Z
Colocation	Uses WALT, fully enabled	Uses WALT, Fully enabled
Freq Aggregation	WALT, aggr. done with timer	WALT, aggr. done with timer
New workload detection	Passed to interactive, 75% threshold	Passed to schedutil, 75% threshold
Early Detection	inform gov of max load based on last_wake_ts condition on first 10 CFS tasks on RQ	Identical impl. to Zone planned
Schedboost	Userspace sets boost-type	Same implementation as Zone, boost types etc.
IRQ Load detection	Threshold set to 10ms	Threshold set to 10ms

EAS-Z TUNABLES

Tunable description	Location	initial value
sched_cpu_high_irqload threshold past which a CPU is marked as having high irq-load	/proc/sys/kernel	10ms
TBA: up/down migration thresholds threshold past up/down {group} migrations are allowed	/proc/sys/kernel	(TBA)

TRACEPOINTS

Sched Trace	Zone	QC EAS
Candidate CPU load	sched_cpu_load_wakeup	sched_cpu_util
Energy Aware	sched_task_load	a) sched_group_energy b) sched_task_util_energy_aware
Bias to Previous CPU	sched_task_load	a) sched_energy_diff b) sched_task_util_energy_diff
Colocation	sched_task_load	sched_task_util_colocated
Bias to Waker Group	sched_task_load	sched_task_util_bias_to_waker
Bias to Waker CPU	sched_task_load	sched_task_util_bias_to_waker
Wake to Idle	sched_task_load	a) sched_group_energy b) sched_task_util_energy_aware
CPU Overutilization	N/A	sched_task_util_overutilized

[Slide shamelessly plagiarized from Daniel Lee (QCT CE Perf)]