softprise
CONSULTING OÜ

Doing big.LITTLE right: little and big obstacles

Uladzislau Rezki, Vitaly Wool
Softprise Consulting OÜ 2015

What is big.LITTLE?

- Complex multicore CPU architecture combining...
  - Several high performance “big” cores
  - Several lower power “small” cores
- Cores should be architecturally compatible
- Cores may be...
  - Of 2 different architectures
  - Of the same architecture but with different...
    - Highest frequency
    - Cache size

Why big.LITTLE?

- Targeting optimal power saving/performance balance
  - Real life CPU load is bursty
    - big.LITTLE allows for running power hungry cores only when bursts are coming
  - Peak performance only when it's needed
  - Power optimized cores run most of the time
- More options for fine tuning compared to standard SMP

Big / LITTLE cores: how to combine

- Clustered switching
  - A cluster of big cores and a cluster of little ones
  - The OS can only use one cluster at a time
  - Standard SMP scheduling within the cluster
- In-kernel switching (CPU migration)
  - Little and big cores are split into pairs
    - Only one core in a pair can be active
  - Standard SMP scheduling within the set of pairs
- Heterogeneous switching (HMP)
  - All cores can be used simultaneously

softprise
CONSULTING OÜ

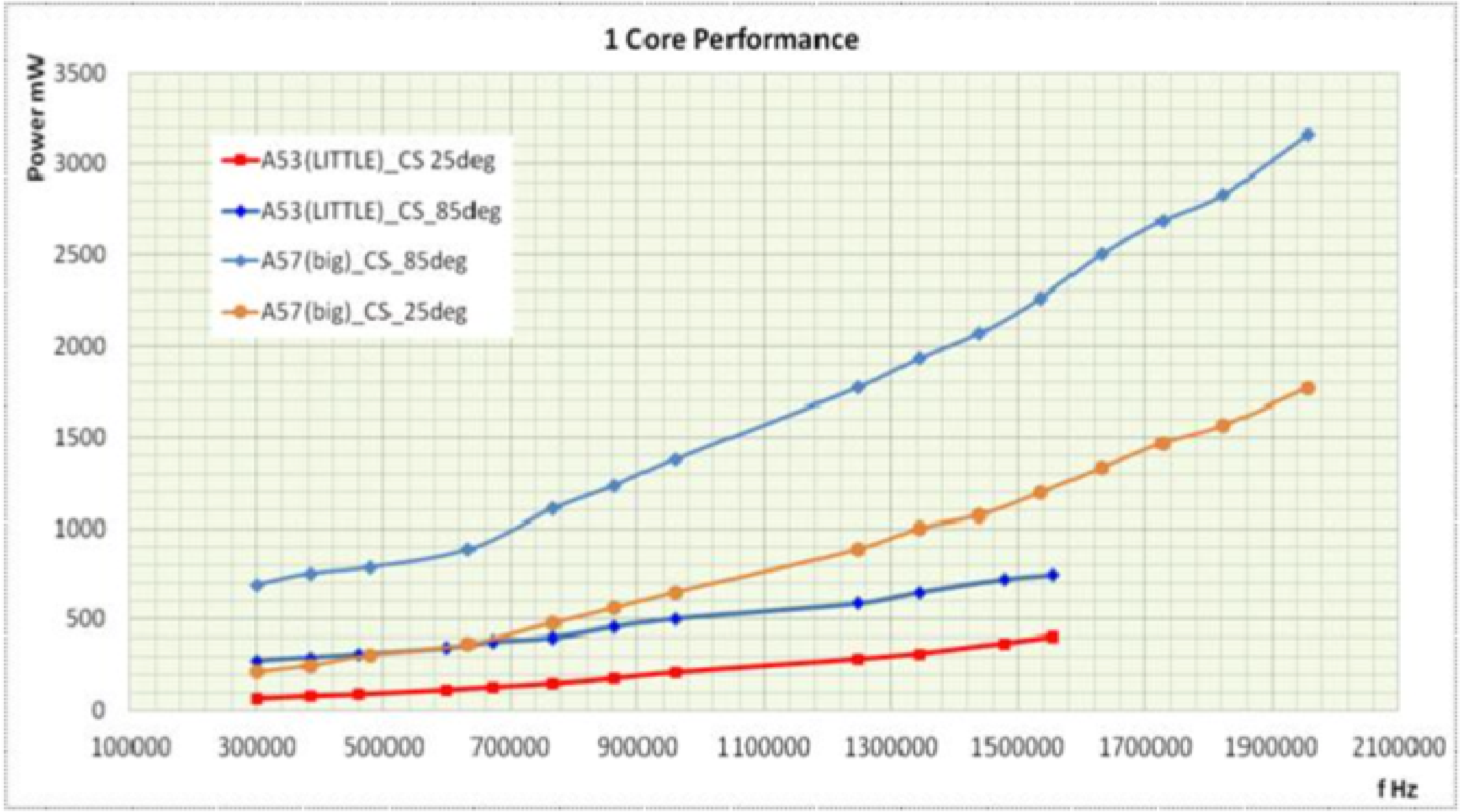
Mainline Linux scheduler ("fair")

- Goals of the fair (CFS) scheduler
  - Even distribution of task load across cores
  - The task ready to run should quickly find core to run on
- Implementation
  - Sorting tasks in ascending order by CPU bandwidth received
    - Red-black trees are used to streamline the process
    - The leftmost task off the tree is picked up next
      - It has the least spent execution time
- Limitations
  - Implies that the cores are the same (e. g. SMP)

“fair” scheduler and big.LITTLE

- Symmetry principle doesn't work well
  - Treating big and little cores as symmetrical is very inefficient
  - Treating tasks as symmetrical doesn't work well too
    - Running big cores is a stress for the system
    - Only really important tasks should run on big cores
- Big cores should be utilized for short time periods
  - And only for “big” tasks
- Scheduler changes required for HMP
  - No consensus in mainline
  - Two competing implementations
    - Qualcomm/Codeaurora vs Linaro/ARM

Performance/power graphs



softprise
CONSULTING OÜ

Big (and LITTLE) obstacles

- Mainline CFS is not really applicable to b.L
  - Global symmetry principle doesn't work in asymmetrical system
- Big cores require careful treatment
  - Should only be run when it's **really** needed
    - Power consumption and heating issues
  - Detection of such situation is the problem to solve
- Task packing problem
- Load balancing problem
  - Covered later in the slides, too

HMP scheduler principles

- Need to account for b.L core differences
- Tasks should be differentiated
  - big/little
  - important/unimportant
- Task scheduling should depend on its properties
  - Task “size” (load-based)
    - Should be calculated somehow
  - Task importance
    - Based on *nice* Linux priorities
      - Not so fine-grained in Android case

softprise
CONSULTING OÜ

Task load calculation

- History window-based load tracking
  - History update events
    - Task starts up/begins execution
    - Task stops execution
  - Demand calculation $$task-demand := \frac{delta \cdot freq_{cur}}{freq_{max}}$$
    - <*delta*>: measure of task's CPU occupancy
    - <$freq_{cur}$>: current frequency of the core
    - <$freq_{max}$>: maximum possible frequency across all cores
  - We should account for core performance
    - Task demand is scaled according to its core's performance

softprise
CONSULTING OÜ

Figuring runnable average (Linaro)

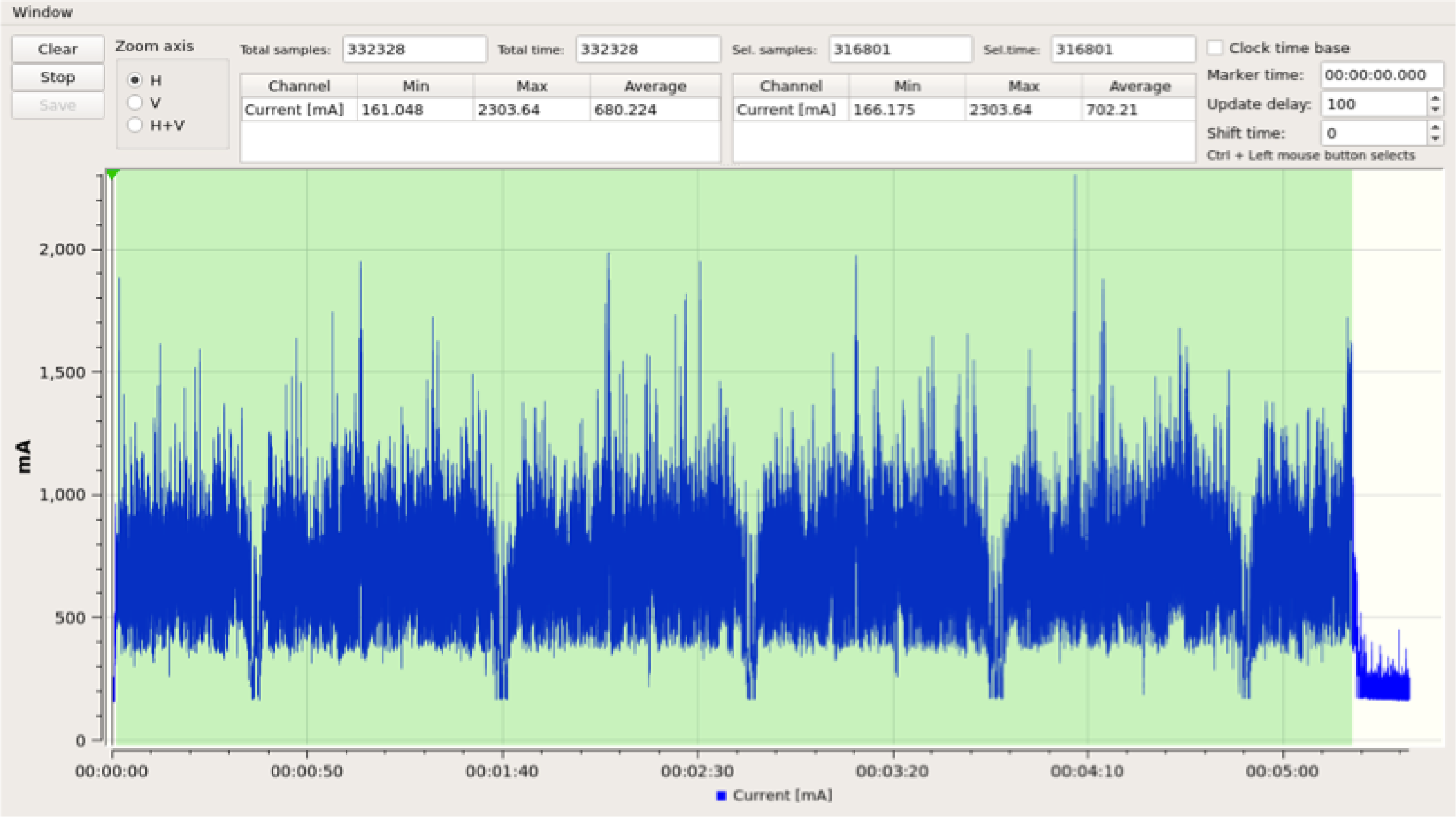
- Runnable history is divided into ~1ms periods
- Weighted load calculation $load := u_0 + u_1 \cdot y + u_2 \cdot y^2 + \ldots$
  - Where $y^{32} = 0.5$
- Advantages of the approach
  - More samples should give better precision
- Some inefficiency detected
  - Computationally heavy
  - Denominator *y* is not easily configurable
    - Load decay is too slow

Window-based load tracking (QC)

- Keeps track of *N* windows of execution per task
  - *N*=5 and *sched_ravg_window*=10 (ms)
- *demand* is calculated as max/average of samples
- Both are extremely power inefficient
  - High spikes when using “max” strategy
  - Slow ramp down when using “average”
  - “hybrid” strategy combines the drawbacks of both
- Our suggestion: weighted load
  - Sample value exponentially decreased over time
  - Bigger *N* gives better precision

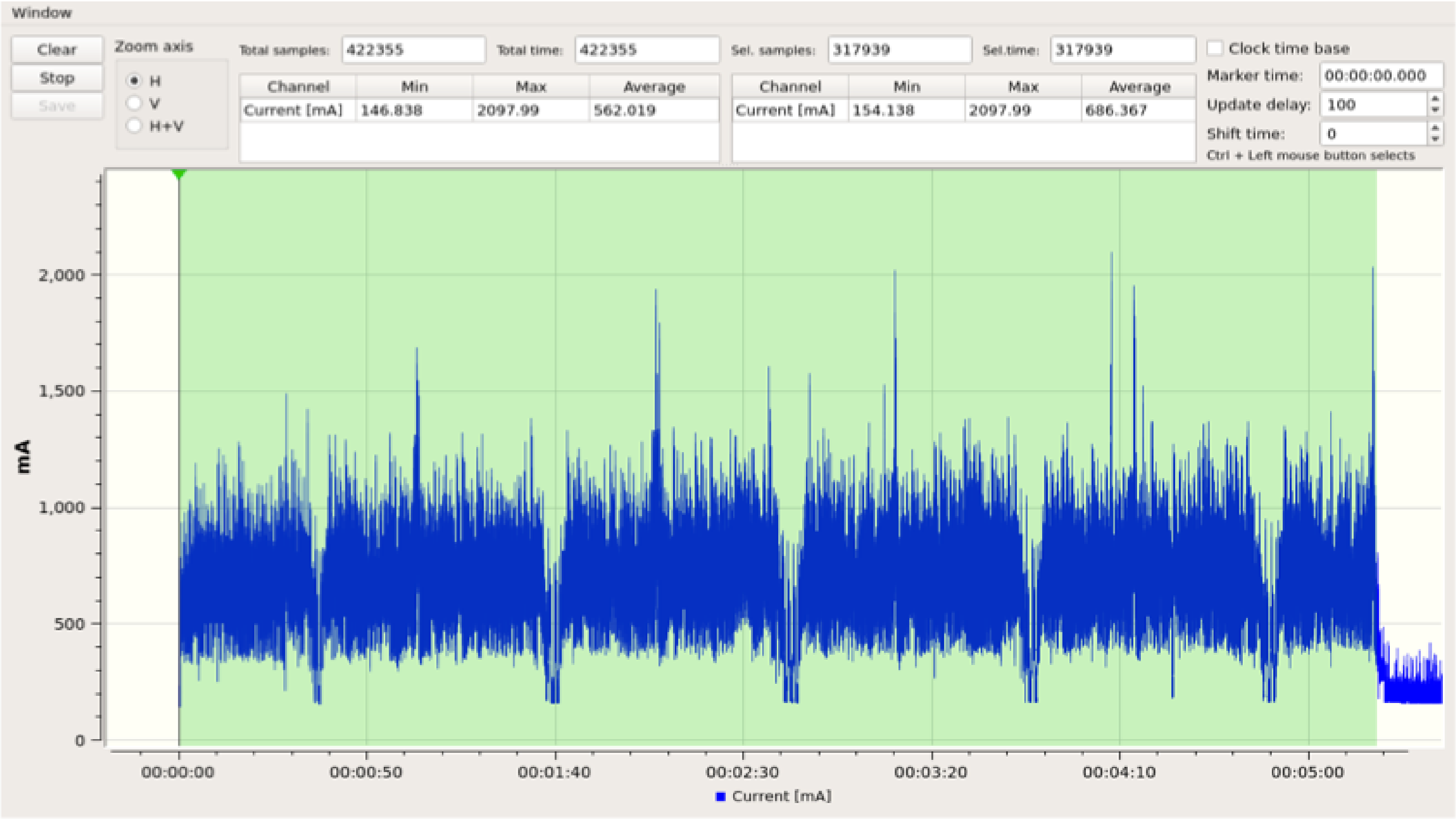
softprise
CONSULTING OÜ

Load tracking: max/avg



softprise
CONSULTING OÜ

Load tracking: exponential WA



“Small” and “big” tasks

- Small task
  - A periodic task with short execution time
  - Can be easily identified using task average demand
    - a task is small if its load is below specified threshold
    - Requires load tracking on scheduler level
- Big task
  - Task producing high CPU load (normally 90%+)
  - Some heavy tasks we don't want to count as big
    - e. g. background threads in Android case
- Not all tasks are either big or small
- Tasks can change their “size” over time

Packing small tasks

- Why pack?
  - Small tasks disturb cores with frequent wake-ups
  - “packing” tasks minimizes wake-ups of different cores, should thus minimize power consumption
- OTOH, packing may result in overloading a CPU
- Packing should be parametrized to allow for fine tuning
  - Depending on the type of application
  - Depending on the architecture of cores
- Implementations differ a lot

Packing: Qualcomm/Codeaurora

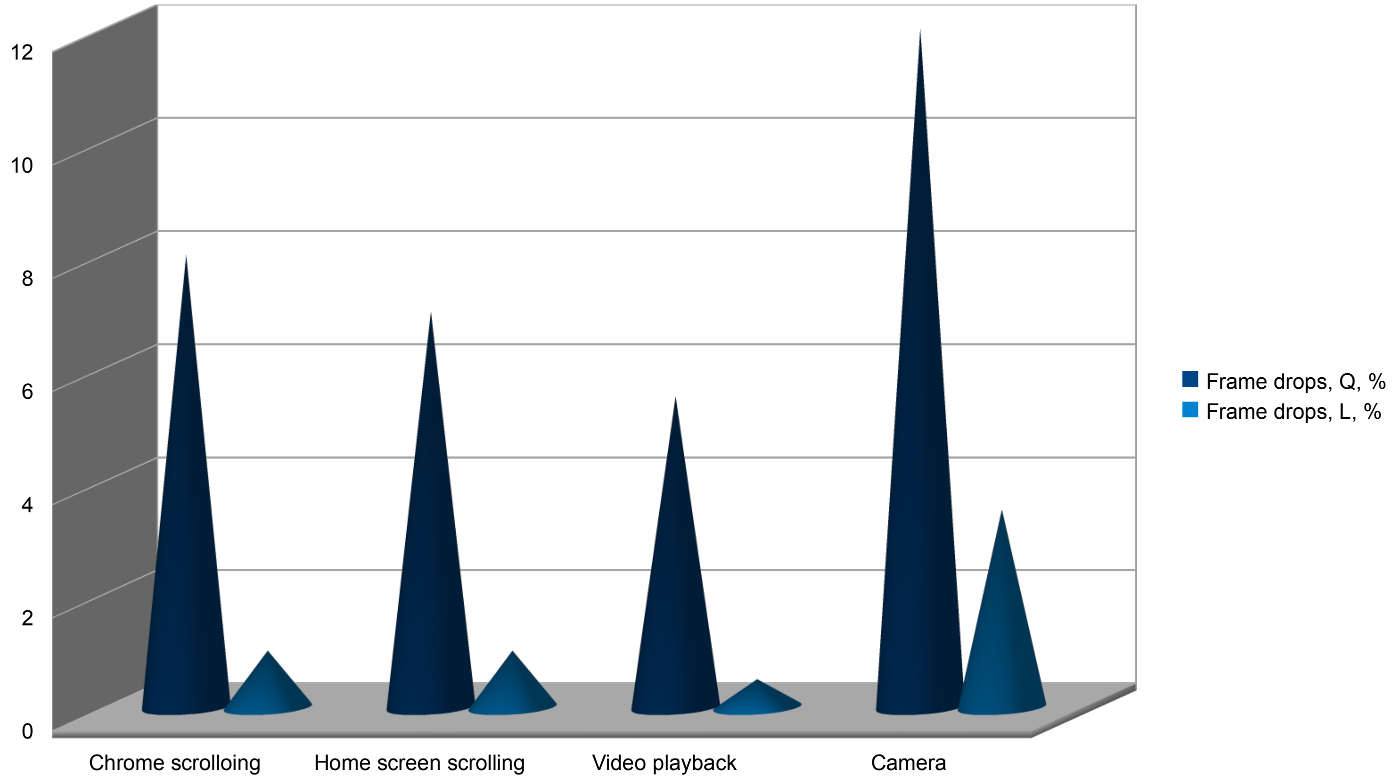
- /sys/devices/system/cpu/cpuX/sched_mostly_idle_freq
- /sys/devices/system/cpu/cpuX/sched_mostly_idle_nr_run
  - A core is considered mostly idle if its frequency and number of running tasks are below respective thresholds
- /sys/devices/system/cpu/cpuX/sched_mostly_idle_load
  - Scheduler will not try to pack tasks from this core if the load is above this threshold
- Seems to give a lot of granularity
  - These parameters are per-core
- Ends up packing all tasks on CPU#0
  - Higher interrupt thread latencies
  - CPU#0 “starvation” possible

Packing: Linaro/ARM

- /sys/kernel/hmp/packing_limit
  - Do not pack tasks on a core if its load will be above this limit **after packing**
- /sys/kernel/hmp/packing_enable
  - Toggle packing process
- Less granular than Qualcomm's implementation
  - No per-core parametrization
- Better behavior in real life scenarios
  - Will not pack everything to a single core for a bursty load

softprise
CONSULTING OÜ

QoS and packing: comparison

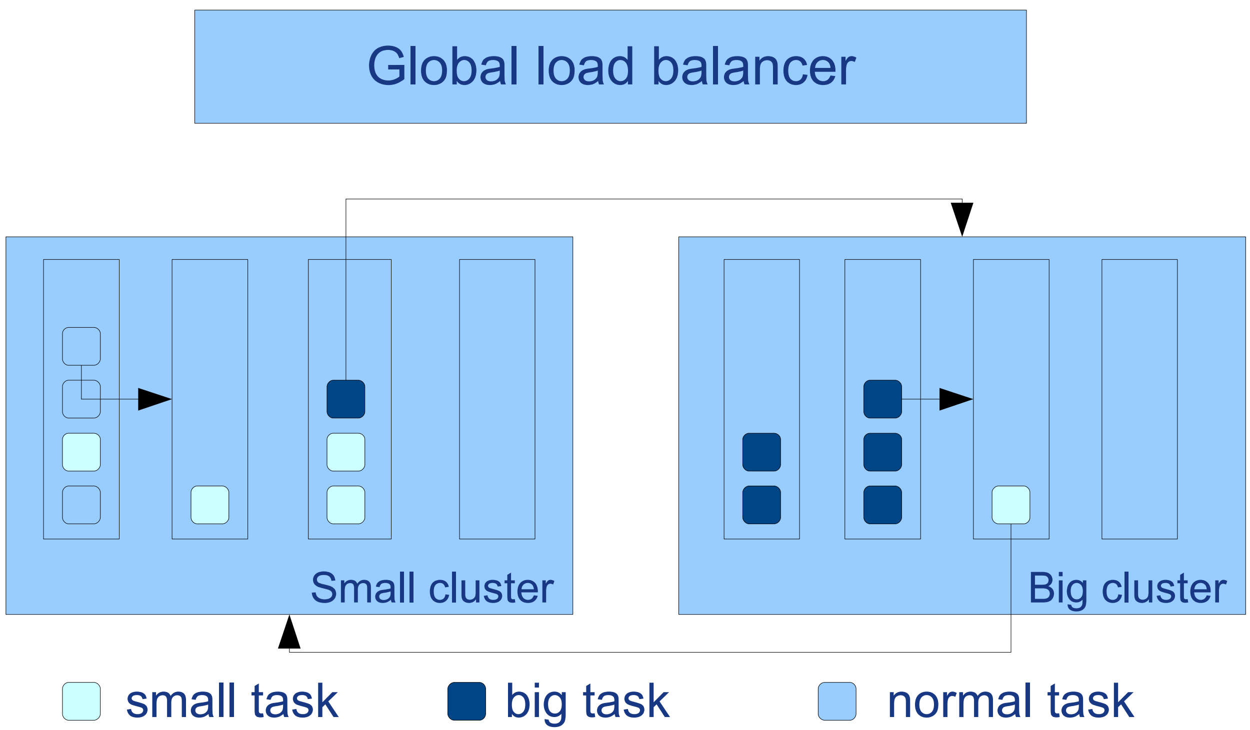


Load balancing

- Runs both per-cluster and per-core
  - Per-cluster balancing pulls tasks between clusters
  - Per-core balancing spreads tasks within cluster
- Algorithm
  - Find the busiest group
  - In this group, find the busiest run queue (CPU)
  - Move tasks from that CPU to another if appropriate
- May conflict with small tasks packing

softprise
CONSULTING OÜ

Load balancing



Refining big tasks selection

- Heavy background tasks are not desired to run on big cluster
  - Compromise the power consumption benefit
  - Or limit the performance gain
- 'Nice' priority based selection is the first step
  - Discount big tasks which have bigger nice value

Android big tasks selection specifics

- Android API defines few nice values for userspace applications
  - Most Android tasks have nice priority 0
  - Discounting these will hurt user experience
- Refine big tasks selection for Android
  - Cgroup-based selection
    - Refuse upmigation for background cgroup tasks

HMP scheduler and CPUfreq

- Objectives
  - HMP scheduler calculates loads anyway
    - It's more efficient to drive/hint CPUFreq from scheduler
    - CPUFreq governor may query scheduler for load
  - CPUFreq can only run within a cluster
  - Scheduler should notify CPUFreq if task is migrated across clusters
- Consequences
  - CPUFreq governors should have HMP support to be used in big.LITTLE systems

Conclusions

- big.LITTLE is a complex architecture that allows for optimizing both power and performance
- Mainline Linux kernel can not leverage well the advantages of big.LITTLE yet
- big.LITTLE kernel support impacts many subsystems
- Leveraging big.LITTLE architecture in Android requires a lot of fine tuning
- big.LITTLE best practices are to be identified yet

softprise
CONSULTING OÜ

Thanks for your attention!

Questions?

mailto:vlad.rezki@softprise.net
mailto: vitaly.wool@softprise.net