Literature Review

of

HMP Scheduling

- big.LITTLE technology

"big" cores should provide maximal computing power

"LITTLE" cores are designed for maximum power efficiency

Peter Hu

Huawei Technologies Research & Development

CPU Team

19 June 2023

HUAWEI

Table of Contents with *references*

Existing Linux CFS and EAS implementation

Ref: EAS Overview and Integration Guide, arm, 2018
https://docs.kernel.org/scheduler

Qualcomm HMP scheduler

Ref: EAS – Energy Aware Scheduler An unbiased look, Vitaly Wool, Konsulko Group

Multimedia, IEEE & Samsung Electronics

saves system-wide (not just CPU) energy consumption by 8.9 percent

Ref: Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-Core Processors (2017)

Abstraction *A*llocates multimedia applications to the small cores and non-multimedia applications to the big cores.

WAEAS, TSINGHUA SCIENCE AND TECHNOLOGY

improves 5-8% when the workload was high

Ref: An Optimization Scheme of EAS Scheduler for Wearable Applications (2021)

BES-WALT Basic exponential smoothing-based WALT algorithm,

Energy-aware CPU selection algorithm,

Abstraction Adjusts *Sched Tune*
Latency-Sensitive Task: decrease searching speed to get minimum performance capacity and the lowest idle state
Non-Latency-Sensitive Task: not choose a lower utilization, but "task packing strategy" (on the little cores)

Batch processing strategy,

Abstraction Adjusts *select idle sibling()*
Record and centralize on fewer cores

Cluster-based load balancing (overutilized)

Abstraction Reduces the meaningless task migration by local load balancing
(when having many aperiodic high-load applications)

Table of Contents with *references*

Learning EAS, LG Electronics *improves power consumption by 2.8% - 7.8%*

Ref: Performance Improvement of Linux CPU Scheduler Using Policy Gradient Reinforcement Learning for Android Smartphones (2020)

Abstraction: Adjusts the *TARGET_LOAD* used to set the CPU frequency and the *sched migration cost* used as the task migration criteria Using Policy reinforcement learning dealing with **workload** or the **ratio of sleep and running states** changes.

Other *references*

Wider coverage: Energy-Aware Scheduling for High-Performance Computing Systems: A Survey (2023)

Existing Linux **CFS** and EAS implementation

Aim:

Performance.
Fairness

Terminology

• Completely Fair Scheduler
models an "ideal, precise multi-tasking CPU" on real hardware.

- CFS maintains the amount of time provided to a given task to determine if it needs balancing
  - the smaller amount of time a task has been permitted access to the processor — the higher its need for the processor is
- CFS maintains a time-ordered red-black tree, sorting tasks in ascending order by CPU bandwidth received
  - Instead of run queues as did predecessors
  - Guarantees O(log(*N*))
- The leftmost task off the red-black tree is picked up next
  - It has the least spent execution time
  - So that task gets the CPU to restore balance (fairness)
- Considers all CPUs to be the same
  - Works very well in SMP systems
  - Does not work in more complicated cases

Source: EAS – Energy Aware Scheduler *An unbiased look*, Vitaly Wool, Konsulko Group

Original CFS policy operates at **full** system utilization,
while EAS operates when the system has low/medium total utilization,
as EAS offers no energy benefit when all CPUs are overutilized

https://docs.kernel.org/scheduler/sched-design-CFS.html
https://docs.kernel.org/scheduler/sched-energy.html

EAS Overview and Integration Guide, arm, 2018

Existing Linux CFS and **EAS** implementation

Aim: **Energy-Efficiency and Performance.**
Fairness

Terminology

- Energy = [joule] (resource like a battery on powered devices)
- Power = energy/time = [joule/second] = [watt]
- CPU Capacity = work_per_hz(cpu) * max_freq(cpu)

Millions of Instructions Per Second (MIPS)

*the **performance** a CPU can reach, normalized against the most performant CPU in the system.*
Heterogeneous systems have asymmetric CPU capacity.

The goal of EAS is to minimize energy, while still getting the job done.
maximize: *performance [inst/s] / Power [W]*

minimize: *Energy [J] / instruction*

EAS:
To break the tie between several good CPU candidates and pick the one that is predicted to yield the **best energy consumption** without harming the system's throughput when deciding which a task should run during wake-up.

Rely on specific elements about the platform's topology, the 'capacity' of CPUs, and their respective energy costs.

https://docs.kernel.org/scheduler/sched-design-CFS.html
https://docs.kernel.org/scheduler/sched-energy.html

EAS Overview and Integration Guide, arm, 2018

Device Tree Source File / Energy Model

```
energy-costs {
        CPU_COST_0: core-cost0 {
                busy-cost-data = <
                        417   168
                        579   251
                        744   359
                        883   479
                        1024  616
                >;
                idle-cost-data = <
                        15
                        0
                >;
        };
        CPU_COST_1: core-cost1 {
                busy-cost-data = <
                        235 33
                        302 46
                        368 61
                        406 76
                        447 93
                >;
                idle-cost-data = <
                        6
                        0
                >;
        };
```

```
        CLUSTER_COST_0: cluster-cost0 {
                busy-cost-data = <
                        417   24
                        579   32
                        744   43
                        883   49
                        1024  64
                >;
                idle-cost-data = <
                        65
                        24
                >;
        };
        CLUSTER_COST_1: cluster-cost1 {
                busy-cost-data = <
                        235 26
                        303 30
                        368 39
                        406 47
                        447 57
                >;
                idle-cost-data = <
                        56
                        17
                >;
        };
```

EAS Overview and Integration Guide, arm, 2018

Existing Linux CFS and EAS implementation

Terminology

	PELT	WALT
Load tracking	Load is accounted using a geometric series	Load is accounted with a policy that observes past *N* windows
Blocked load/utilization tracking	Load is decayed as part of a runqueue statistic when the task is blocked	Blocked load contribution is removed from runqueue sum/average statistics.
Blocked load restoration	Runqueue statistics include blocked load/utilization at all times	Load contribution is restored to RQ statistics when the task becomes runnable again.

Load Tracking Mechanism

Per- Entity Load Tracking (PELT) - focused only on a single class - CFS

Let *Li* designate the entity's load contribution in period *pi*

- Then the total load is

$$L = L_0 + L_1 q + L_2 q^2 + L_3 q^3 + \cdots$$

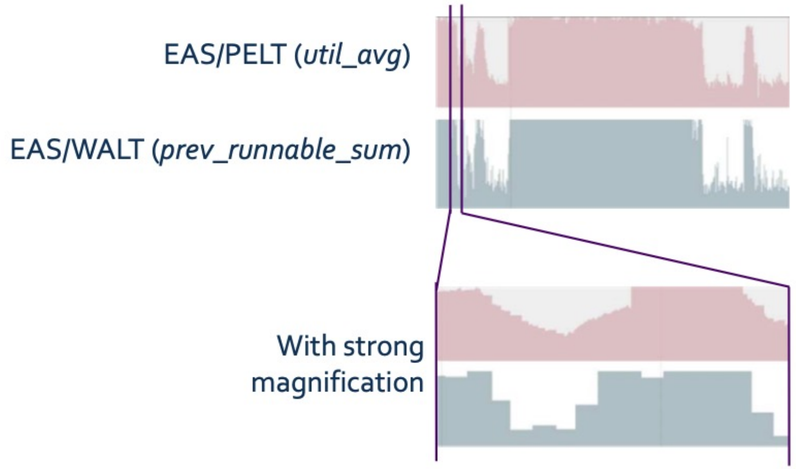
q is the decay factor

Window Assist Load Tracking (WALT) - EAS

faster reaction times when the behaviour of tasks changes.
uses periodic calculations that are synchronized across all of the run queues, attempting to track the behaviour of **all** scheduling classes.

✅ the decisions can be made based on the information about the full state of the running system.

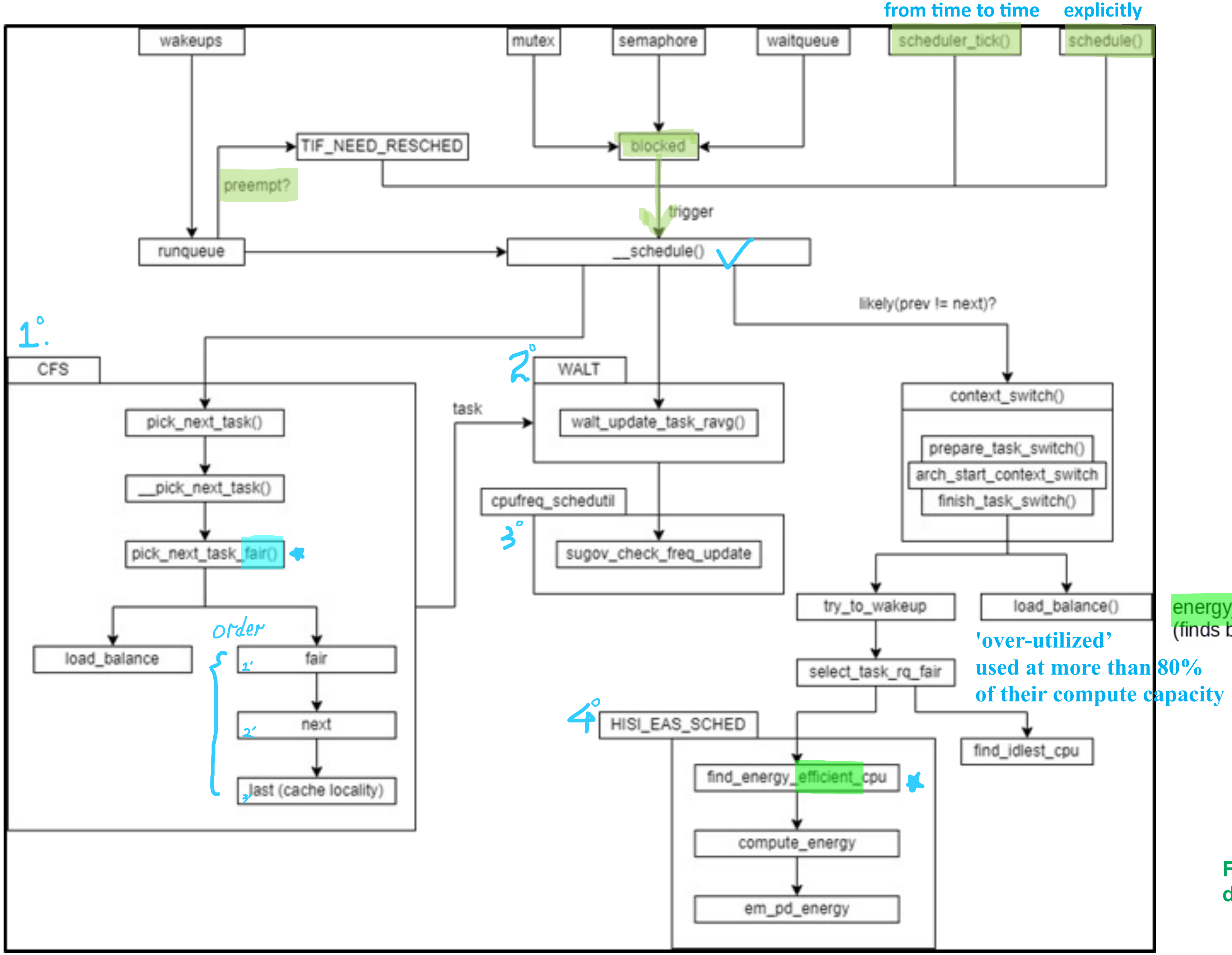
❌ additional locking complexity and some additional delays in other pathways



https://docs.kernel.org/scheduler/sched-design-CFS.html
https://docs.kernel.org/scheduler/sched-energy.html

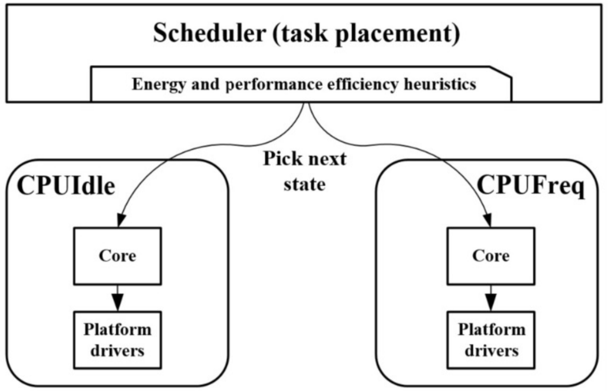
EAS Overview and Integration Guide, arm, 2018

Existing Linux CFS and EAS implementation



CPU capacity

a measure of the **performance** a CPU can reach, normalized against the most performant CPU in the system.



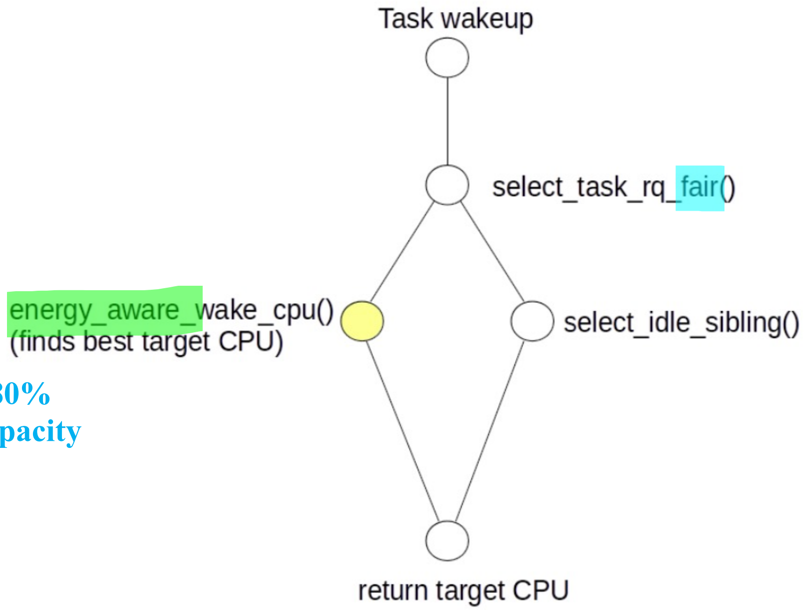


Figure 2 Task wakeup path modifications. (The yellow dots represent EAS functionality)

1.8 EAS Overview

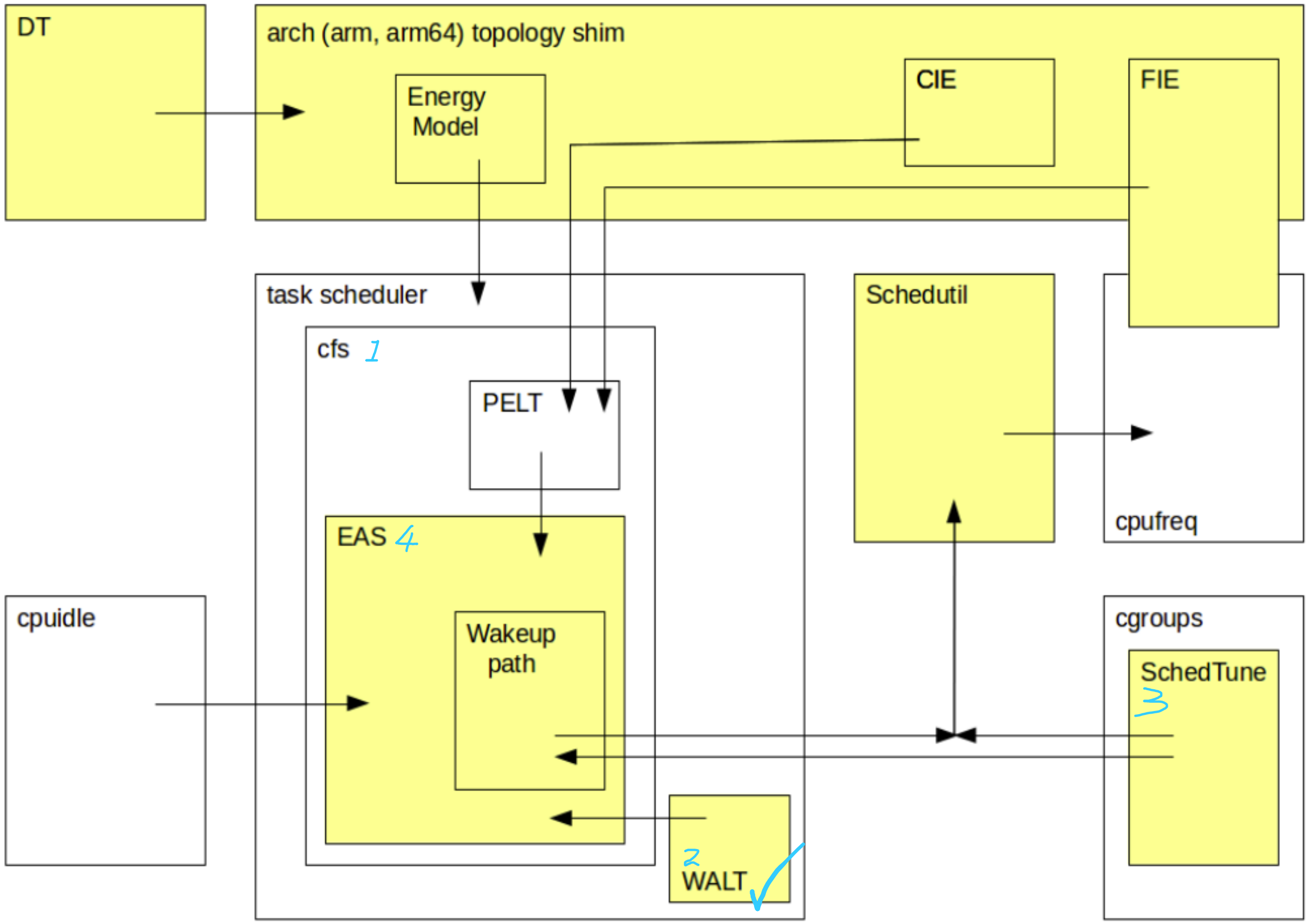


Figure 1 EAS building blocks in relation to Linux task scheduler, cgroups subsystem and related power management subsystems

EAS Overview and Integration Guide, arm, 2018

Sched util - Scheduler driven DVFS

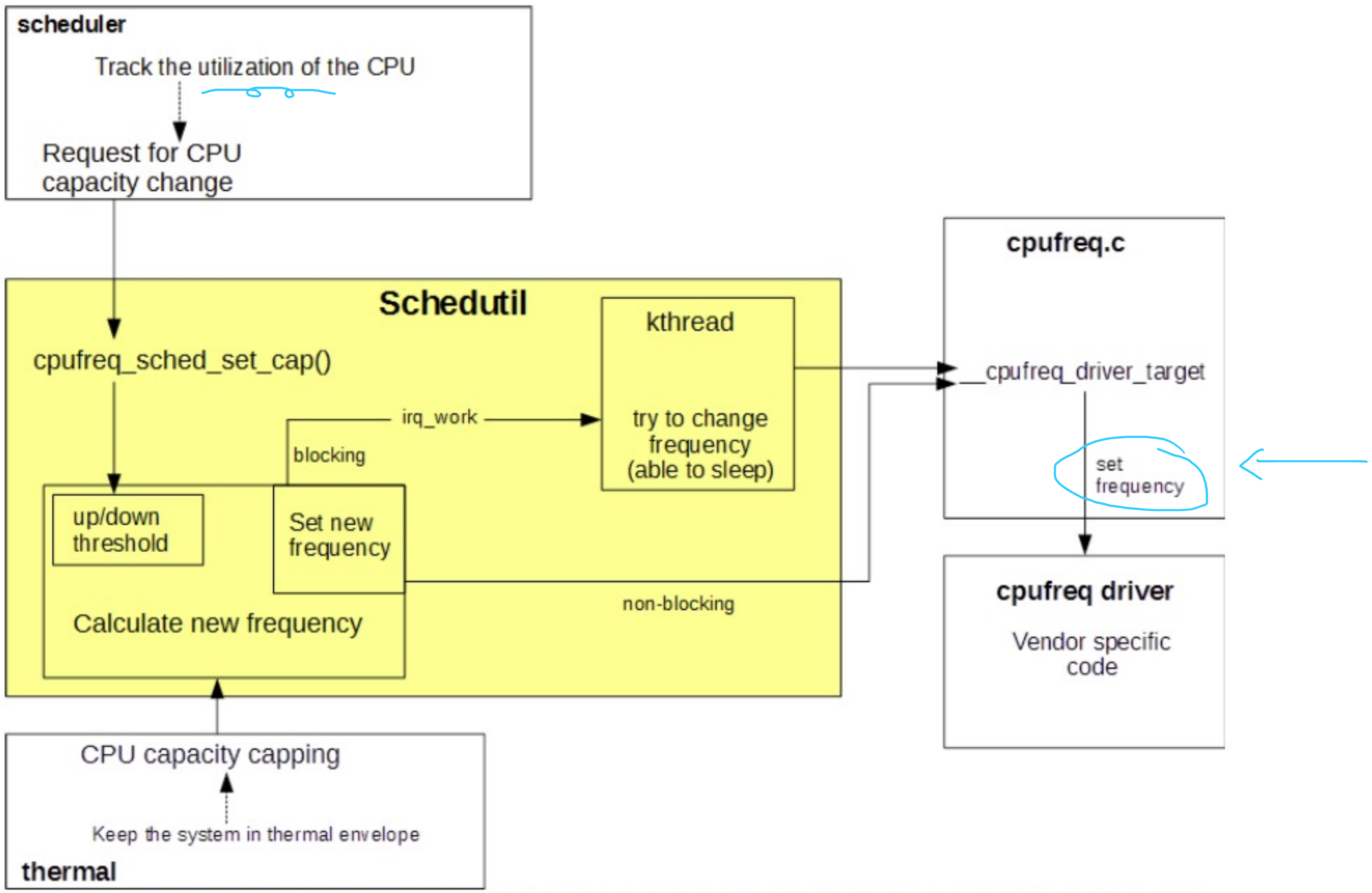


Figure 11 Schedutil block diagram showing connections between scheduler, thermal subsystem and existing CPUFreq

EAS Overview and Integration Guide, arm, 2018

Sched Tune - Task classification and control

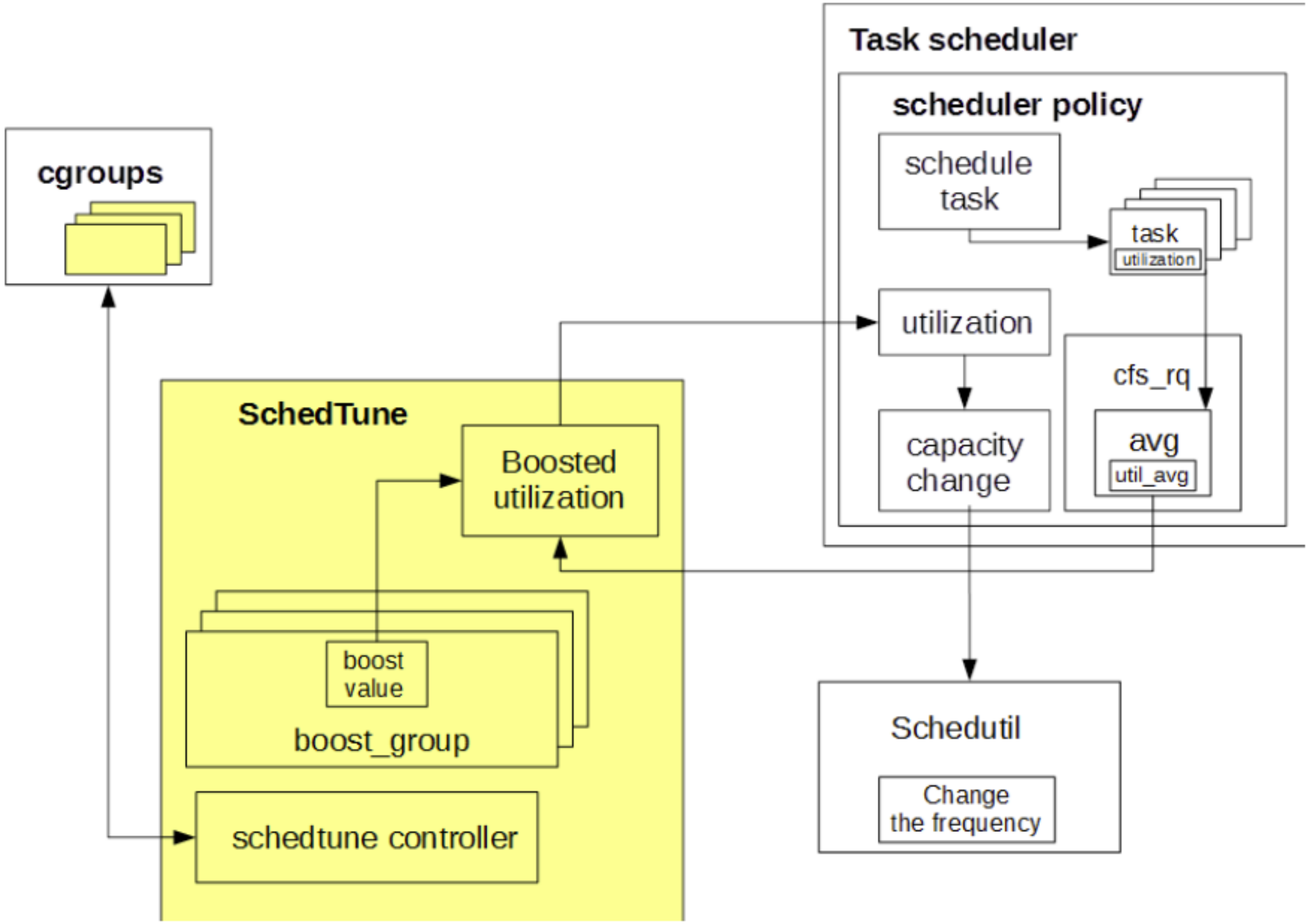


Figure 12 SchedTune block diagram showing components and connections between scheduler policy and boosted group's utilization concept.

This value is used to compute a margin to be added or removed to or from the utilization signal of a task/cpu. The value of the margin is calculated to provide a well-defined and expected user-space behaviour. For example, the following table reports the meaning of some specific boost values:

Boost value [%]	Meaning (e.g. run the task at a frequency corresponding to)
0	Minimum required capacity (max energy efficiency)
100	Maximum possible speed (min time to completion) (*)
50	Something in between the previous two configurations
-50	Half of the minimum required capacity
-100	Minimum available capacity (minimum OPP)

(*) minimum latency is not yet completely supported in the current ACK release, this feature is a work in progress and will be added in a following release.

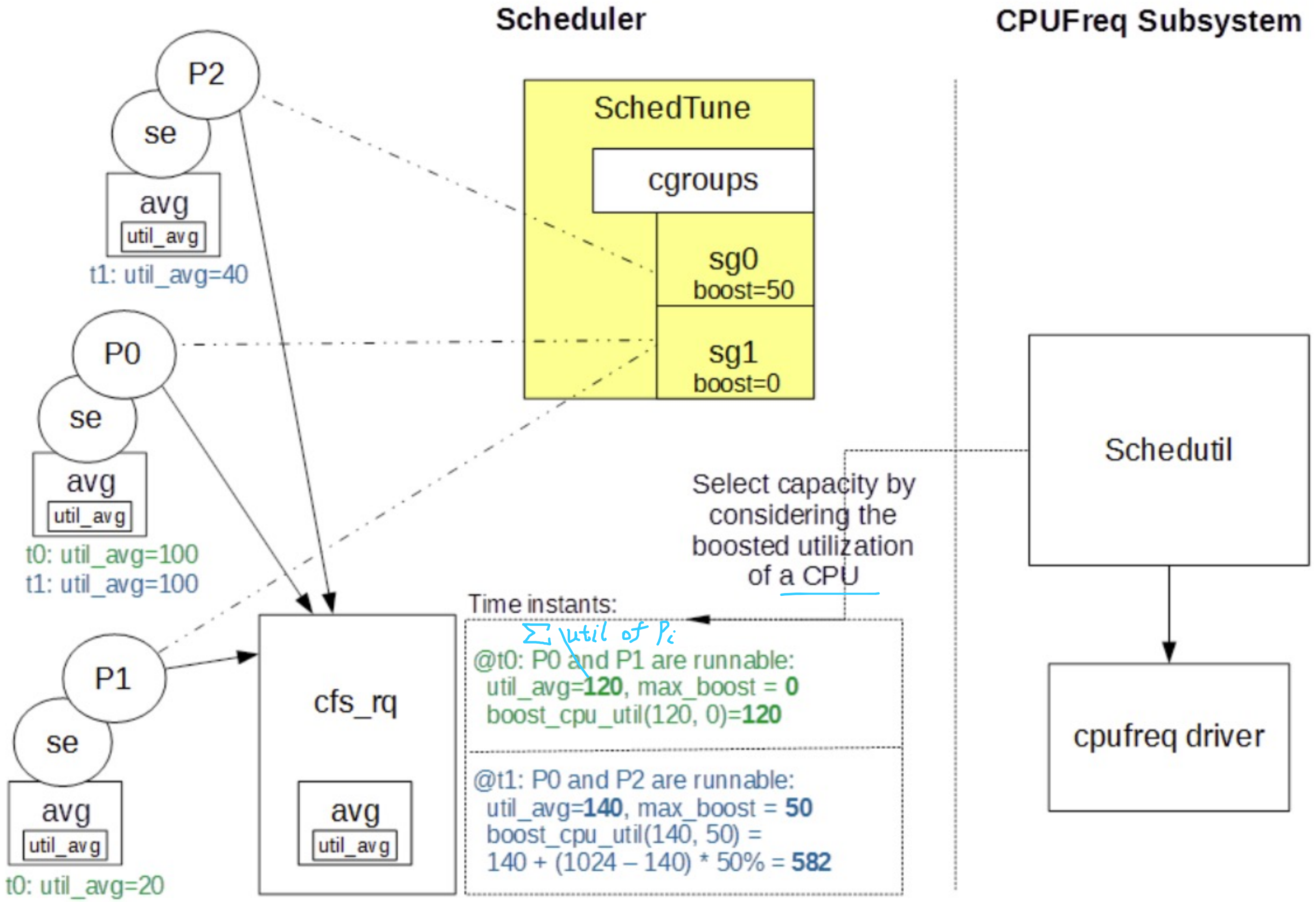


Figure 13 Flow diagram showing the state of the SchedTune and kernel scheduler in time: t0, t1.

Existing Linux CFS and **EAS** implementation

Aim: **Energy-Efficiency and Performance.**
Fairness

Terminology

- Energy = [joule] (resource like a battery on powered devices)
- Power = energy/time = [joule/second] = [watt]

The goal of EAS is to minimize energy, while still getting the job done.

maximize: *performance [inst/s] / Power [W]*

minimize: *Energy [J] / instruction*

EAS:
To break the tie between several good CPU candidates and pick the one that is predicted to yield the **best energy consumption** without harming the system's throughput when deciding which a task should run during wake-up.

Rely on specific elements about the platform's topology, the 'capacity' of CPUs, and their respective energy costs.

https://docs.kernel.org/scheduler/sched-design-CFS.html
https://docs.kernel.org/scheduler/sched-energy.html

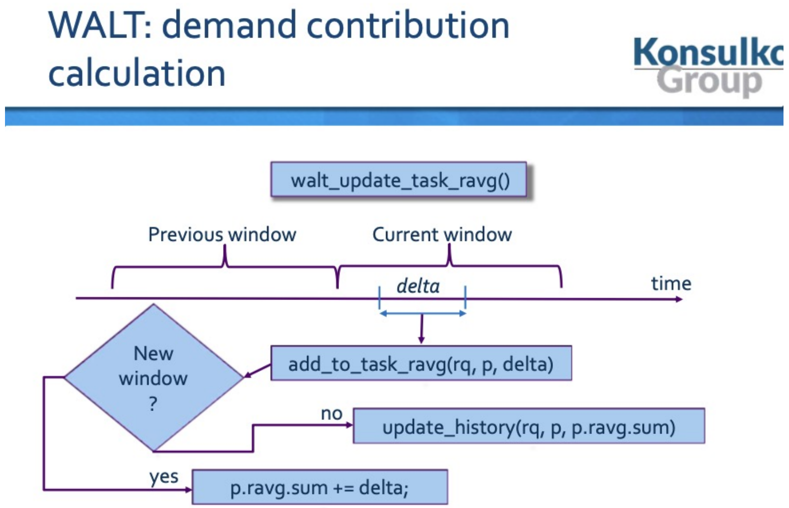
EAS Overview and Integration Guide, arm, 2018

Existing Linux CFS and **EAS** implementation

Qualcomm HMP scheduler

1.25*D -- 80% capcity

- *Task demand* D_{task} is the contribution of a task's running time to a window
  - $D_{task} = \frac{delta\_time \times cur\_freq}{\max\_possible\_freq}$
    - *delta_time* - time of task running on a core in a period of time
    - *cur_freq* - the current frequency of the core this task is running on
    - *max_possible_freq* is the maximum possible frequency across **all** cores
- Calculated over *N* sliding windows (*N* is a parameter)
  - E. g. the average demand $D_{avg} = (D_1 + \cdots + D_N)/N$
  - The best result is achieved with $\mathrm{D} = \max\{D_{avg}, D_1\}$



- We already account for difference in maximum frequency
  - D_{task} is calculated in regard to maximum frequency across all cores
- We also need to account for higher performance of big cores
  - $D_{task,scaled} = D_{task} \cdot \frac{rq \rightarrow efficiency}{\max possible\ efficiency}$
    - *Efficiency* is a per-runqueue parameter
    - Usually big cores are considered 2x more effective

Multimedia

To eliminate the above inefficiency of the conventional task scheduler, we propose an advanced task scheduler for heterogeneous mobile multi-core processors. **Our proposed task scheduler** allocates multimedia applications to the small cores and non-multimedia applications to the big cores.

Since recent smart devices have employed dedicated hardware decoders for multimedia applications

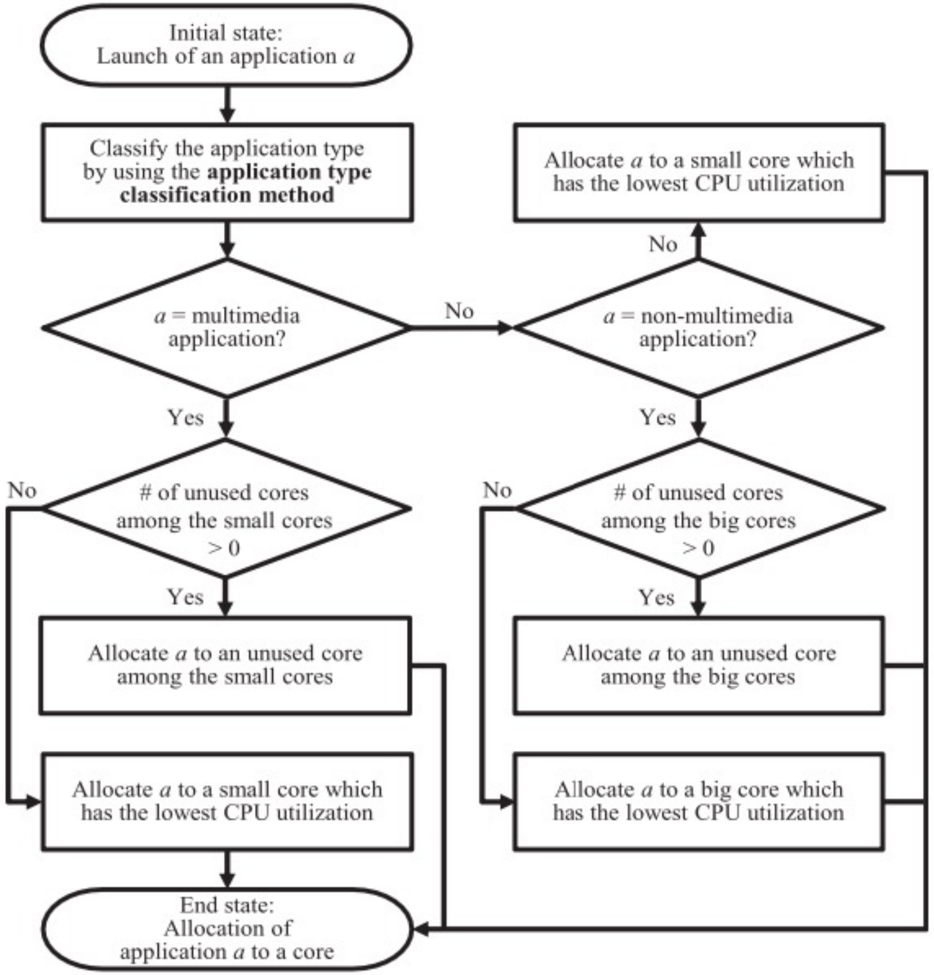


Fig. 5. Allocation algorithm of our proposed task scheduler.

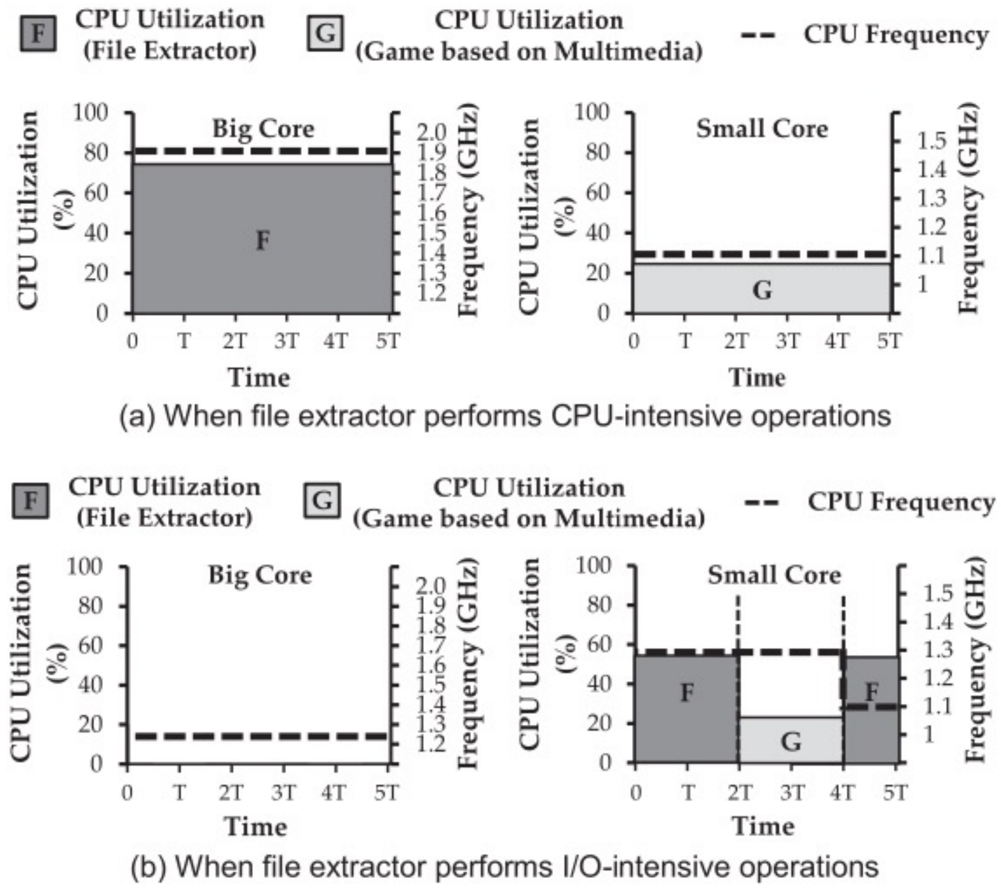


Fig. 2. Example of conventional task scheduler.

Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-Core Processors (2017)

WAEAS

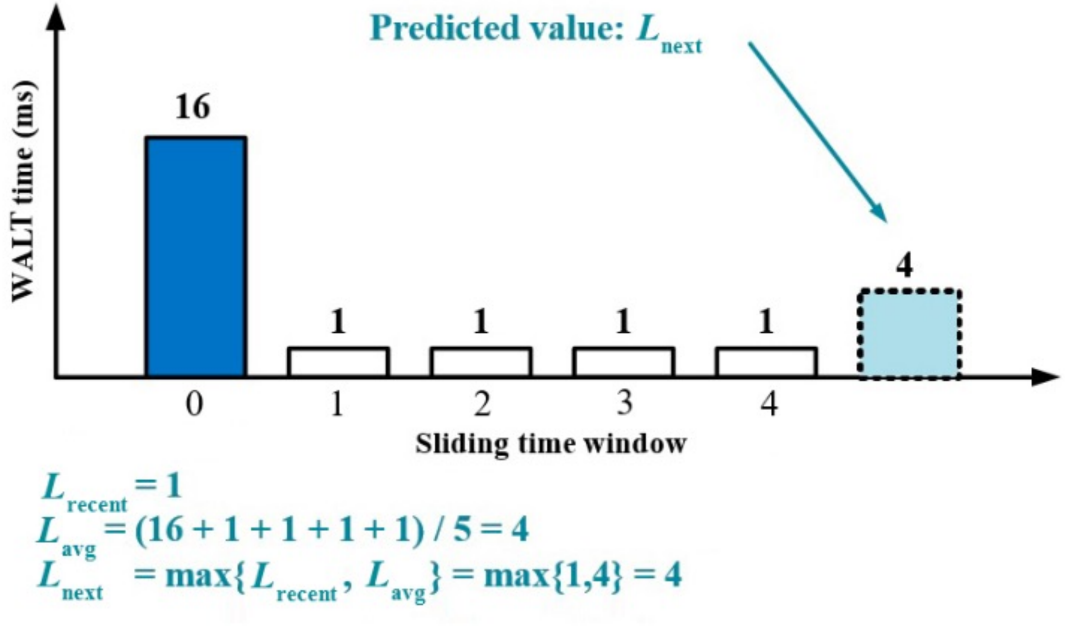
This paper analysed performance energy efficiency in the dynamic scheduling of latency-sensitive and non-latency-sensitive tasks for heterogeneous multicore wearable systems. By aiming to improve the energy efficiency of workload prediction, CPU core selection, and load balancing without affecting performance, the authors further optimized the EAS scheduler and proposed WAEAS, such as

BES-WALT **Basic exponential smoothing-based WALT algorithm,**

$$L = \text{time}_{\text{exec}} \times \frac{\text{freq}_{\text{curr}}}{\text{freq}_{\text{max}}} \times \frac{\text{IPC}_{\text{curr}}}{\text{IPC}_{\text{max}}}$$

$$L_{t+1} = \alpha \times x_t + (1 - \alpha) \times L_t$$

Lt represents the predicted demand in the t-th sliding window,
xt represents the real demand in the t-th sliding window,
represents the smoothing factor, where a is in [0,1]



WAEAS: An Optimization Scheme of EAS Scheduler for Wearable Applications (2021)

WAEAS

This paper analysed performance energy efficiency in the dynamic scheduling of **latency-sensitive and non-latency-sensitive tasks** for heterogeneous multicore wearable systems. By aiming to improve the energy efficiency of workload prediction, CPU core selection, and load balancing without affecting performance, the authors further optimized the EAS scheduler and proposed WAEAS, such as

BES-WALT Basic exponential smoothing-based WALT algorithm,

Energy-aware CPU selection algorithm,

Sched Tune

Latency-Sensitive Task: decrease searching speed to get minimum performance capacity and the lowest idle state

Non-Latency-Sensitive Task: not choose a lower utilization, but “task packing strategy” (on the little cores)

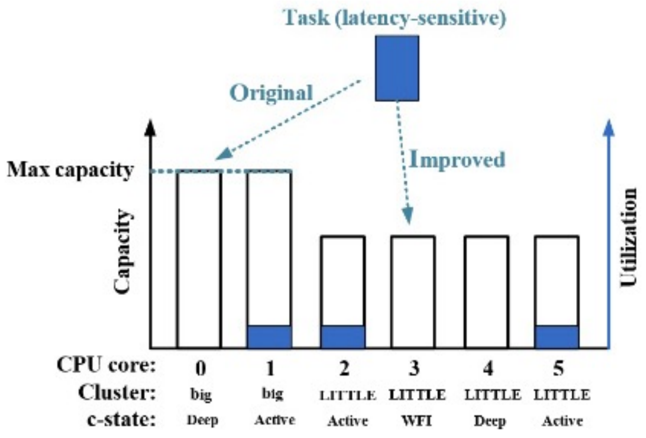


Fig. 5 Example of CPU selection for latency-sensitive tasks.

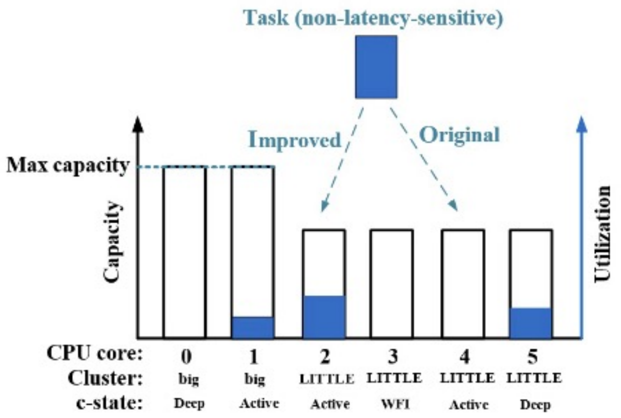


Fig. 6 Example of CPU selection strategy for non-latency-sensitive task.

WAEAS: An Optimization Scheme of EAS Scheduler for Wearable Applications (2021)

WAEAS

This paper analysed performance energy efficiency in the dynamic scheduling of latency-sensitive and non-latency-sensitive tasks for heterogeneous multicore wearable systems. By aiming to improve the energy efficiency of workload prediction, CPU core selection, and load balancing without affecting performance, the authors further optimized the EAS scheduler and proposed WAEAS, such as

BES-WALT Basic exponential smoothing-based WALT algorithm,
Energy-aware CPU selection algorithm,
Batch processing strategy,

select idle sibling()
The proposed strategy records which core the task was running on as the awakener at the last time, and runs the tasks in the system centralized on fewer cores, which not only reduces the migration of tasks in the system, but also makes more cores in the system be in the idle state.

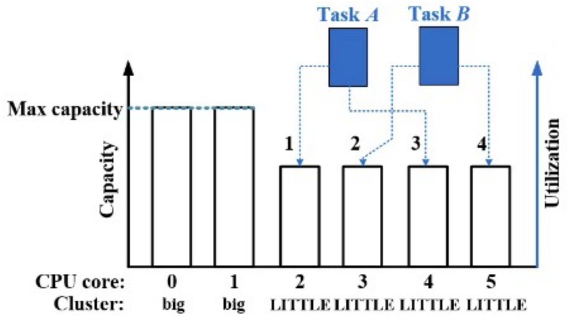


Fig. 7 **Example of Tasks *A* and *B* awaken each other in the original algorithm: Task *A* (Core 2) wake up Task *B* (Core 3); Task *B* (Core 3) wake up Task *A* (Core 4); Task *A* (Core 4) wake up Task *B* (Core 5).**

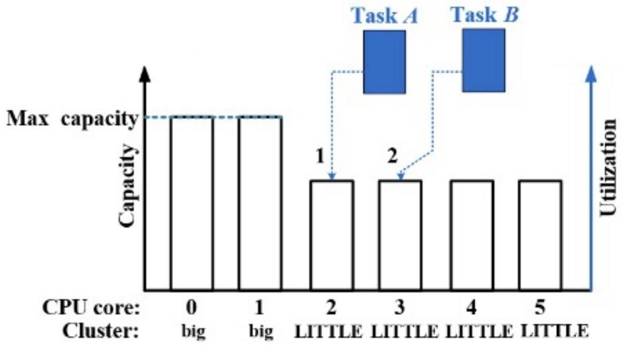


Fig. 8 **Example of Tasks *A* and *B* waking up each other in the optimized algorithm: Task *A* (Core 2) wake up Task *B* (Core 3); Task *B* (Core 3) wake up Task *A* (Core 2); Task *A* (Core 2) wake up Task *B* (Core 3).**

WAEAS: An Optimization Scheme of EAS Scheduler for Wearable Applications (2021)

WAEAS

This paper analysed performance energy efficiency in the dynamic scheduling of latency-sensitive and non-latency-sensitive tasks for heterogeneous multicore wearable systems. By aiming to improve the energy efficiency of workload prediction, CPU core selection, and load balancing without affecting performance, the authors further optimized the EAS scheduler and proposed WAEAS, such as

BES-WALT Basic exponential smoothing-based WALT algorithm,
Energy-aware CPU selection algorithm,
Batch processing strategy,
Cluster-based load balancing (overutilized)

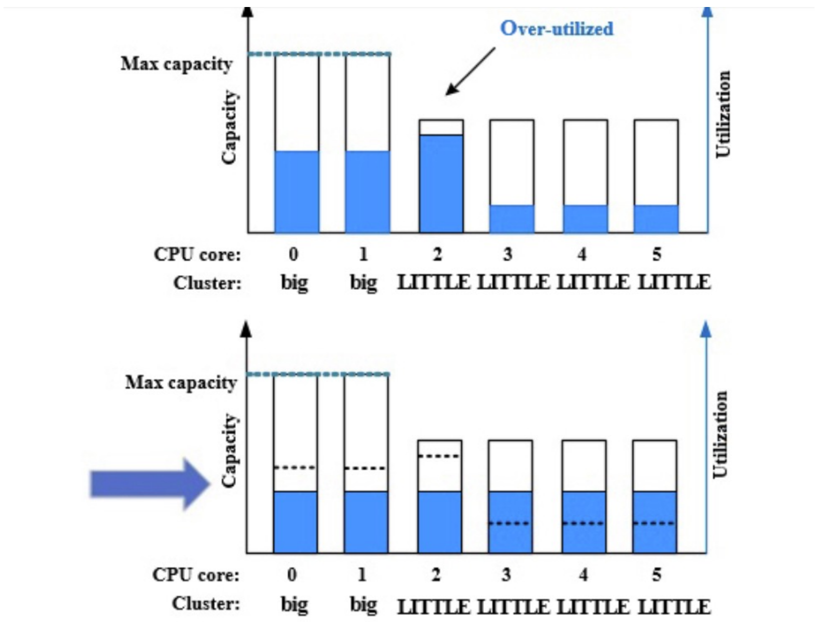


Fig. 9 Example of the original load balancing in EAS scheduler.

reduces the meaningless task migration *by local* load balancing
sets an overutilized indicator in the **big** core scheduling subdomain and the **little** core scheduling subdomain, **respectively**, thereby indicating that whether the domain is overutilized or not. When a scheduling domain is overutilized, the current domain state is set to be overutilized first, and the load balancing in the current domain is given a priority

aperiodic high-load applications

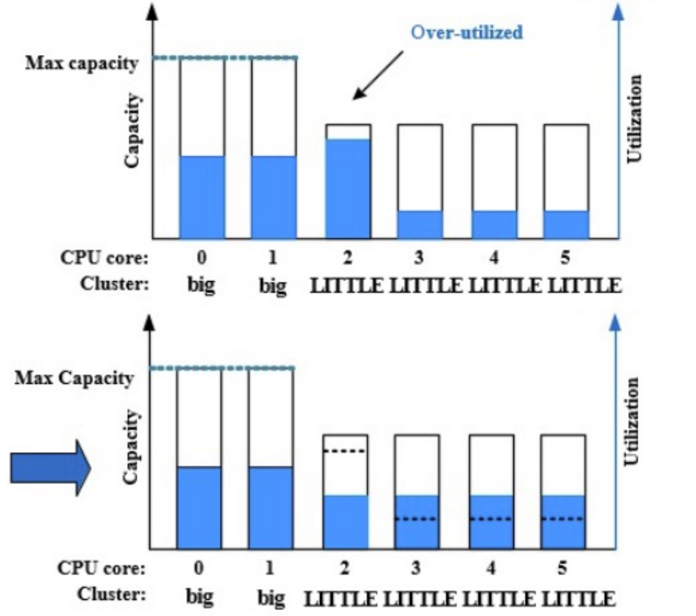


Fig. 10 Example of the cluster-based load balancing strategy.

WAEAS: An Optimization Scheme of EAS Scheduler for Wearable Applications (2021)

Learning EAS

The Learning EAS adjusts the TARGET_LOAD used to set the CPU frequency and the sched_migration_cost used as the task migration criteria according to the characteristics of the running task through the **policy gradient reinforcement learning**.

Effects:

In LG G8 ThinQ, Learning EAS improved power consumption by 2.3% - 5.7%, hackbench results for process scheduling performance by 2.8% - 25.5%, applications entry time by 4.4% - 6.1%, and applications entry time under high CPU workload by 9.6% - 12.5%, respectively compared with EAS. This paper also showed that the Learning EAS is scalable by applying the Learning EAS to high- end and low-end chipset platforms of Qualcomm.Inc and MediaTek.Inc and improving power consumption by 2.8% - 7.8%, application entry time by 2.2% - 7.2%, respectively compared with EAS. Finally, this paper showed that the performance of CPU scheduling is improved gradually by the repetition of reinforcement learning.

Performance Improvement of Linux CPU Scheduler Using Policy Gradient Reinforcement Learning for Android Smartphones (2020)

Learning EAS

Motivation:

EAS generally provides good performance because many of the values which are basis for the operation are **fixed.**
In general, each task has its own characteristics for changing *workload*.
It also has unique properties for *transition between running state and sleep state*.
It is difficult to optimize the performance by using simple algorithm which changes the default value for the EAS operation when a task's ***workload*** or the ***ratio of sleep and running states*** gets to change above or below a certain value.

Solution:

Set the values of the EAS operation criteria dynamically according to the characteristics of the running tasks to improve performance, which can be achieved through **policy gradient reinforcement learning**.

updating parameter vector

$$\theta_{t+1} = \theta_t + \alpha \frac{\partial \rho}{\partial \theta_t}$$

ρ policy performance measure

α learning rate

Performance Improvement of Linux CPU Scheduler Using Policy Gradient Reinforcement Learning for Android Smartphones (2020)

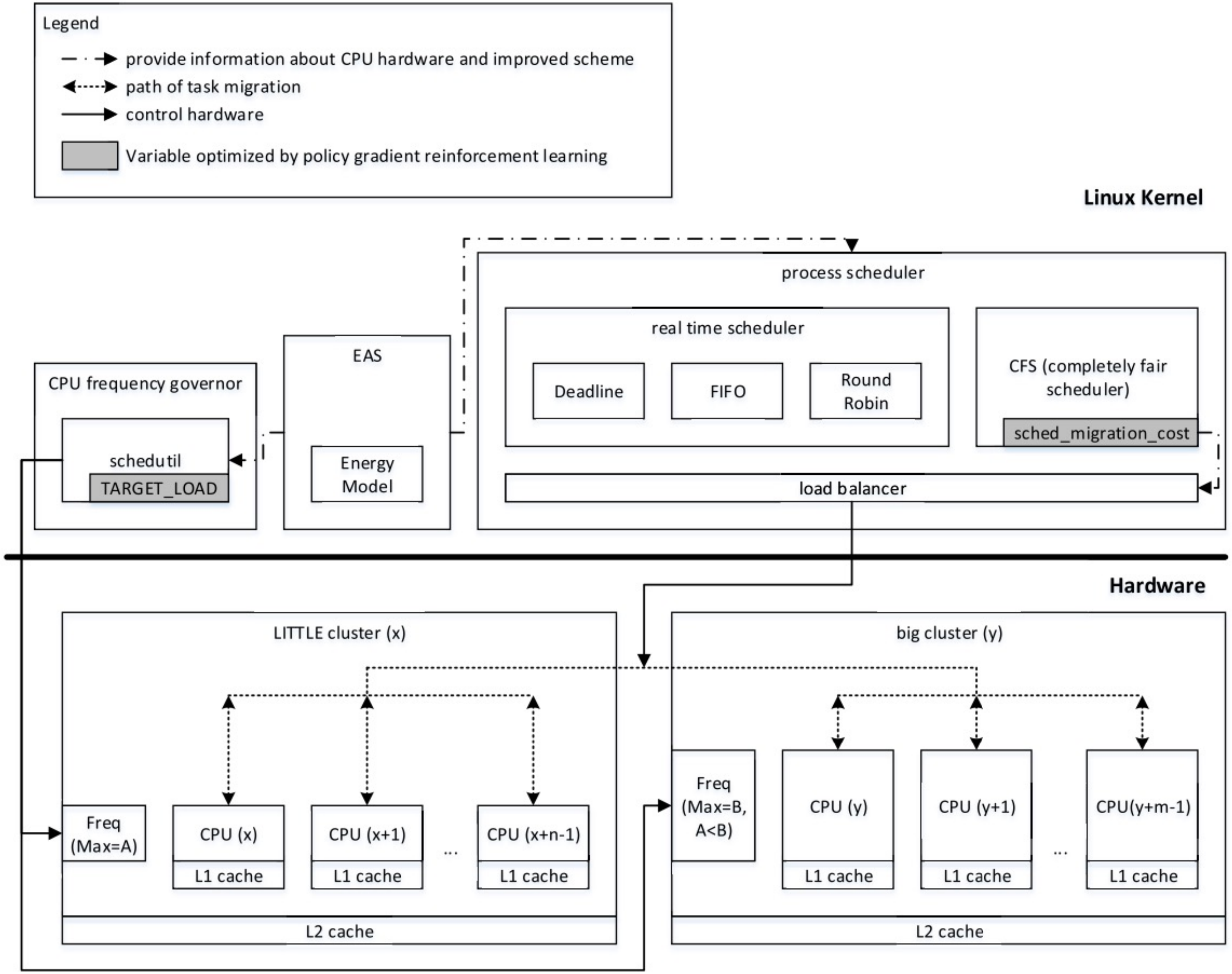


FIGURE 3. Block diagram of EAS and big.LITTLE chipset platform.

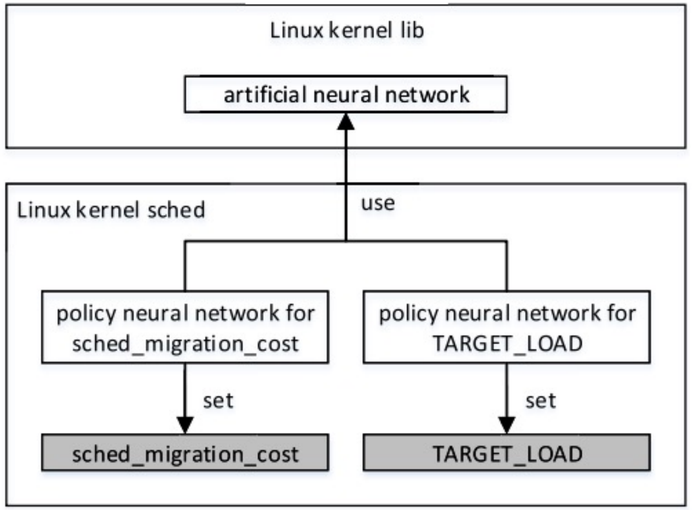


FIGURE 7. Diagram of policy gradient reinforcement learning for learning EAS.

cy Gradient Reinforcement Learning

Learning EAS

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

$\int \pi(s)$

$\pi(a_0, s_0)$ Q

$$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} \dots \xrightarrow{a_{T-1}^{(1)}} s_T^{(1)}$$
$$s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} \dots \xrightarrow{a_{T-1}^{(2)}} s_T^{(2)}$$
$$\dots$$
$$s_0^{(m)} \xrightarrow{a_0^{(m)}} s_1^{(m)} \xrightarrow{a_1^{(m)}} s_2^{(m)} \xrightarrow{a_2^{(m)}} \dots \xrightarrow{a_{T-1}^{(m)}} s_T^{(m)}$$

State *s* each CPU util

Action a increase or decrease TARGET LOAD

Policy π : S × A → [0,1] $\pi(a, s) = \Pr(a_t = a \mid s_t = s)$

the agent's action selection, π mapping from the states to the actions.

Stationary Probability $d^\pi(s) = \lim_{t\to\infty} P(s_t = s | s_0, \pi_\theta)$

State-value function V : S × π → R $V^\pi(s) = E[R \mid s, \pi]$

Action-value function fw/Q: S × π → R Expected return

Performance Function ρ $\rho(\theta) = \rho^{\pi_\theta}$, θ space → policies πθ space

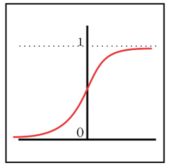
parameter θ the policy that *increases or decreases* TARGET_LOAD which affects the next CPU frequency change.

Action Preference Function h(s, a, theta)

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

Sigmoid / SoftMax

Input $X \in \mathbb{R}$ ⟶



⟶ **Output** $P(Y=k|X) \in [0,1]$

Optimal Value Function

$$\min_{T_j} \sum_{j=1}^{Ncpu} P(T_j) \quad P(T_j) \approx a \cdot (T_j)^2 + b$$

$$\textit{subject to} \sum_{i=1}^{Ntask} W_i \leq \sum_{j=1}^{Ncpu} T_j \qquad (6)$$

CPU workload ≤ CPU throughput

In schedutil, each CPU workload is expressed as

$$Cutil = \left(\frac{Ctime}{Ctime + Itime}\right) \cdot MCutil$$

Max

largest CPU util in the cluster

Next

$$NCfreq \cdot (0.8) = \left(\frac{LCutil}{MCutil}\right) \cdot MCfreq$$

$$NCfreq = \left(\frac{LCutil}{MCutil}\right) \cdot MCfreq \cdot (1.25)$$

Reward function

$$J(\theta) = \sum_{s\in\mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a)$$

Direct policy search:

Updating parameter vector

$$\theta_{t+1} = \theta_t + \alpha \frac{\partial \rho}{\partial \theta_t}$$

α learning rate

Policy Gradient Theorem

$$\frac{\partial \rho}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s, a)}{\partial \theta} f_w(s, a) \qquad (11)$$

d^π is the stationary distribution of states under π, which is independent of s_0 for all policies, and f_w is approximation of quality function. State s is each CPU util and a is the action to increase or decrease TARGET_LOAD. There are two typical

current TARGET_LOAD change value

$$\frac{\partial \rho}{\partial \theta_t} = tl_t(freq_t - freq_{t-1})$$

Thus when <u>π (*s*, *a*) is partially differentiated against θ</u>, it is the rate of the CPU *<u>frequency change</u>*.

CS229 Ch 16.Reinforcement Learning and Control

https://en.wikipedia.org/wiki/Reinforcement_learning

https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

Performance Improvement of Linux CPU Scheduler Using Policy Gradient Reinforcement Learning for Android Smartphones (2020)

Learning EAS

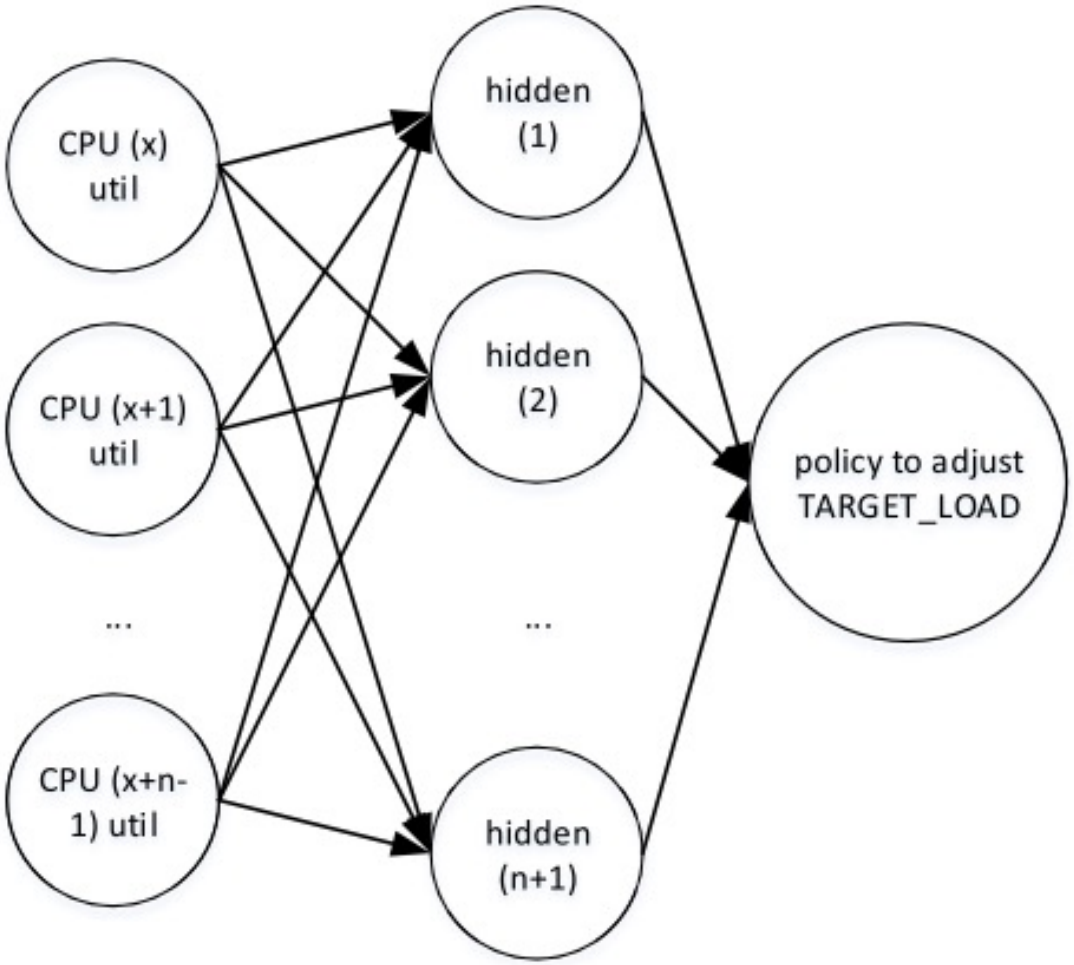


FIGURE 4. Configuration of policy neural network to set TARGET_LOAD for cluster (x).

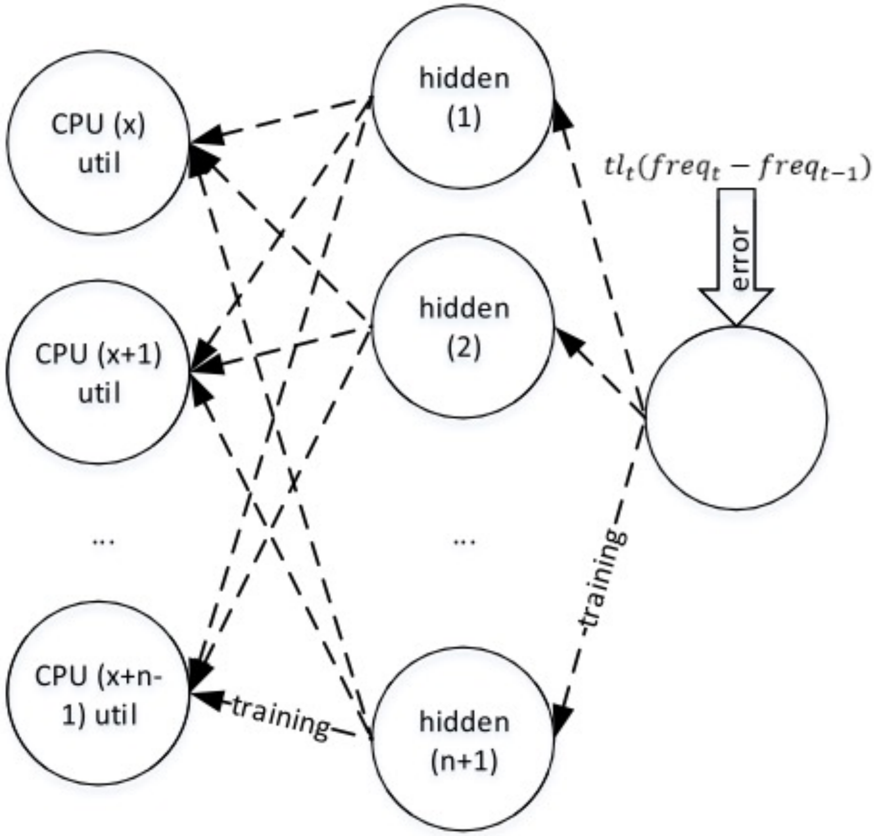


FIGURE 5. Training of policy neural network to set TARGET_LOAD for cluster (x).

$$\frac{\partial \rho}{\partial \theta_t} = tl_t(freq_t - freq_{t-1})$$

Performance Improvement of Linux CPU Scheduler Using Policy Gradient Reinforcement Learning for Android Smartphones (2020)

Learning EAS

Algorithm 1 Pseudocode to set TARGET_LOAD for Cluster (x) Using Policy Gradient Reinforcement Learning

Initialize policy neural network
$CPU\_utils[n] = \{0, 0, \ldots, 0\}$
$prev\_CPU\_freq = cur\_CPU\_freq = 0$
repeat
  $i = 0$
  while $i < n$ **do**
    $CPU\_utils[i]$ = Get i*th* CPU load
    $++i$
  end
  $result$ = Run policy neural network(CPU_utils)
  if ($threshold(12000) < result$) **then**
    tl = -0.0023
  else then
    tl = 0.0023
  end
  TARGET_LOAD $+=$ tl
  $prev\_CPU\_freq = cur\_CPU\_freq$
  Sleep 100ms
  Get cur_CPU_freq
  $error = tl * (cur\_CPU\_freq - prev\_CPU\_freq)$
  Training policy neural network by back propagation($error$)
until system has been shutdown

Algorithm 2 Pseudocode to set Sched_Migration_Cost Using Policy Gradient Reinforcement Learning

Initialize policy neural network
$CPU\_load\_vari[m] = \{0, 0, \ldots, 0\}$
$prev\_total\_CPU\_load\_vari = total\_CPU\_load\_vari = 0$
$mc = 0$
$prev\_mig\_count = mig\_coun = 0$
$prev\_mig\_success = mig\_success = 0$
repeat
  Sleep 7000ms
  $total\_CPU\_load\_vari = 0$
  $i = 0$
  while $i < m$ **do**
    $CPU\_load\_vari[i]$ = Get the variance of CPU workloads in i*th* cluster
    $total\_CPU\_load\_vari += CPU\_load\_vari[i]$
    $++i$
  end
  mig_count = Get task migration attempts counts
  $mig\_success$ = Get task migration success counts
  $cur\_error = (total\_CPU\_load\_vari / (mig\_count + mig\_success))$
  $prev\_error = (prev\_total\_CPU\_load\_vari / (prev\_mig\_count + prev\_mig\_success))$
  $error = -mc * (cur\_error - prev\_error)$
  Training policy neural network by back propagation($error$)
  $result$ = Run policy neural network (mig_count, $mig\_success$, $CPU\_load\_variance$)
  if ($threshold(160000) < result$) **then**
    $mc = -17027$
  else then
    $mc = 17027$
  end
  sched_migration_cost $+=$ mc
  $prev\_total\_CPU\_load\_vari = total\_CPU\_load\_vari$
  $prev\_mig\_count = mig\_count$
  $prev\_success\_count = success\_count$
until system has been shutdown

Performance Improvement of Linux CPU Scheduler Using Policy Gradient Reinforcement Learning for Android Smartphones (2020)

Thank you!
Best of Luck~

Peter Hu

Huawei Technologies Research & Development

CPU Team

HUAWEI