CPU Frequency Scaling in Linux

Giovanni Gherdovich
ggherdovich@suse.cz

Before we begin

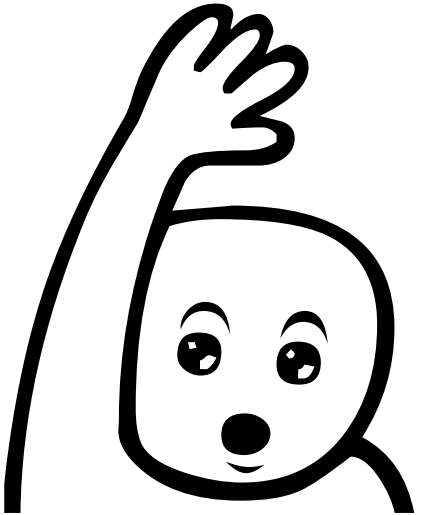
SUSE is hiring!

http://www.suse.com/jobs

Agenda

- > Power management overview
- > Governors, drivers
- > Governors close-up
  - > intel_pstate powersave
  - > generic ondemand
  - > generic schedutil

Questions, anytime



What am I running?

```
$ cpupower --cpu all frequency-info --driver
$ cpupower --cpu all frequency-info --policy
$ cpupower --cpu all frequency-info --governors
```

Active Power Management

Scaling voltage and frequency of the CPUs to consume as little power as needed while guaranteeing acceptable performance

Active Power Management

instantaneous power consumption of a CPU:

$$P \propto C\, V^2\, f + P_s$$

_ C – capacitance
_ V – voltage
_ f – clock frequency
_ P_s – power when halted

- V, f move together
- P is quadratic in V
- P_s is low (since Nehalem)

see https://physics.stackexchange.com/a/34778 for a discussion

Active Power Management

Theodore Ts'o ▸ Public May 16, 2013

+Arjan van de Ven +H. Peter Anvin Some folks internally have been arguing (and with data that appears to support their thesis) that with modern Intel processors, the ondemand CPU governor is actually counterproductive because waking up to decide whether the CPU is idle keeps it from entering the deepest sleep states, and so (somewhat counterintuitively) the performance governor will actually result in the best battery life.

https://plus.google.com/+TheodoreTso/posts/2vEekAsG2QT

Active Power Management

Race to halt – is it enough?

> as long as your idle power is low enough (Nehalem+) ☺

> ADD-type VS LOAD-type instructions ☹

> race-to-halt has a linear effect ($P \propto P_s$) but P is quadratic in voltage ☹

> ACPI specs VS µarch-specific controls ☹

$\Longrightarrow$ intel_pstate driver/governor

P-states a.k.a. Operating Performance Points

Defined by the ACPI spec (Advanced Configuration and Power Interface)

http://www.uefi.org/sites/default/files/resources/ACPI_6_2.pdf

> G-states: global system states (G0...G3)

> D-states: device power states (D0...D3)

> C-states: CPU power states (C0...C3)

> P-states: [device|processor] performance states (P0...Pn)

> S-states: sleeping states (S1...S5)

The lower the state number, the “more active” a component is.

P-states a.k.a. Operating Performance Points

> a pair (V, f) determining the CPU operating state
> each (logical!) core can be at a different P-state
> V, f grow together
  > the higher the values, the higher the performance...
  > ...but also the power consumption

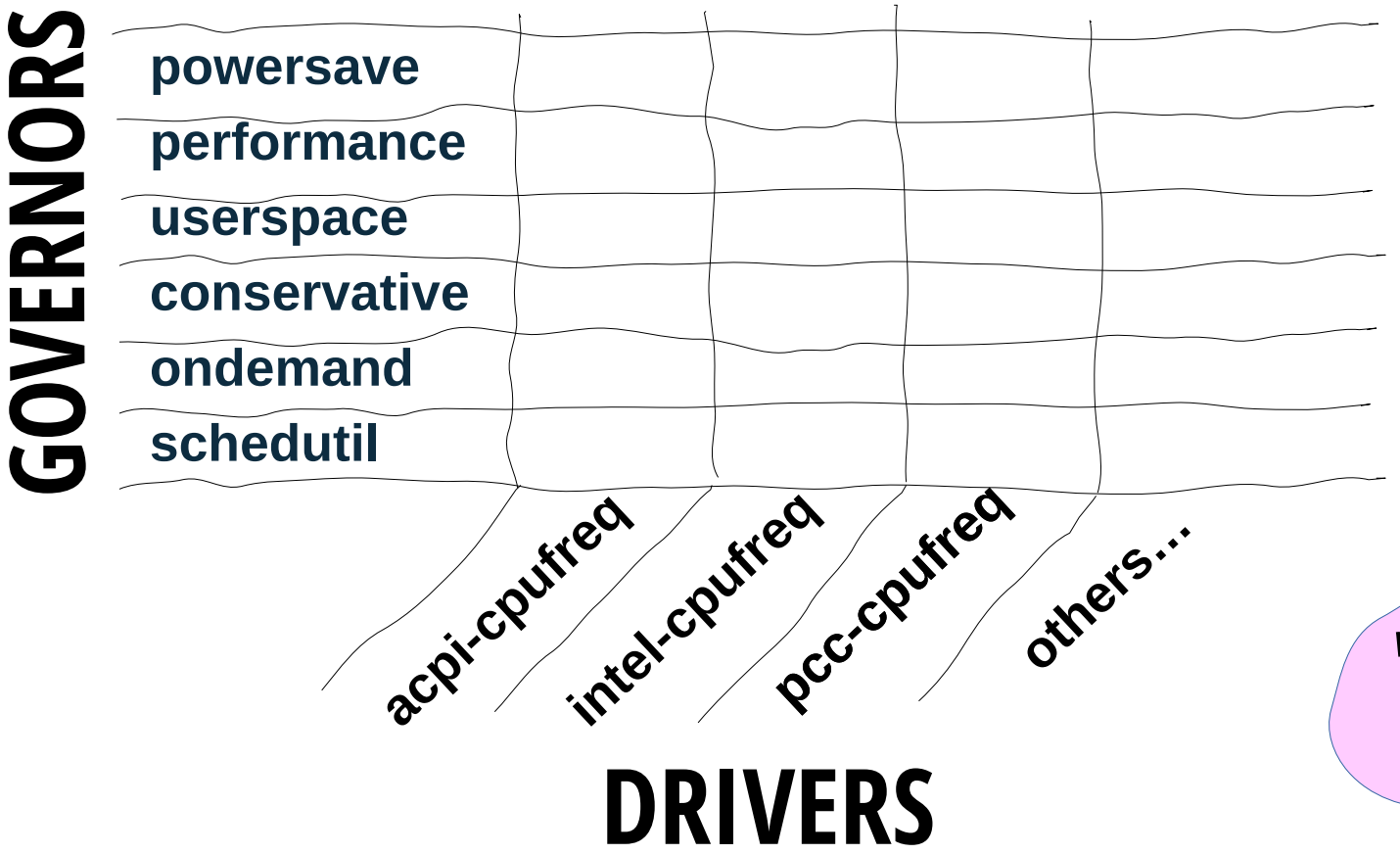
Governors and Drivers

> governor: decides if change is necessary / which frequency to go next

> driver: applies the change to CPUs

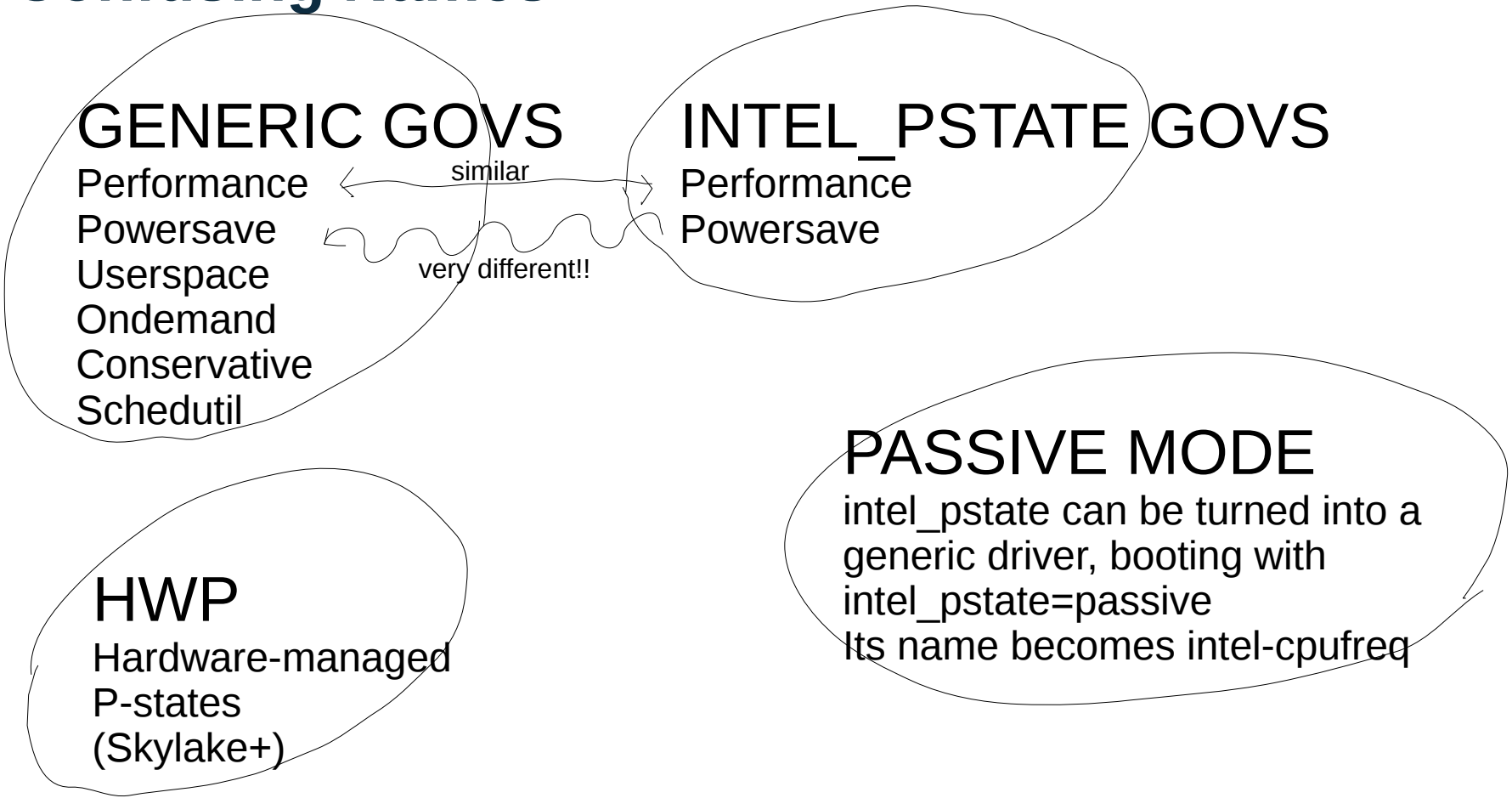
CPUFreq was designed to make governors and drivers separate from each other

Choose your driver, choose your governor



not intel_pstate!
It embeds its own governors

Confusing Names



Governors close-up

intel_pstate powersave governor (since Linux v3.9)

> input:

> registers APERF, MPERF (model-specific)

intel_pstate powersave governor

> idea

> Proportional Integral Derivative (PID) controller

e(t) error at time t

$$\text{correction}(t) = A\, e(t) + B \int_0^t e(s)ds + C\, d/dt\, e(t)$$

proportional

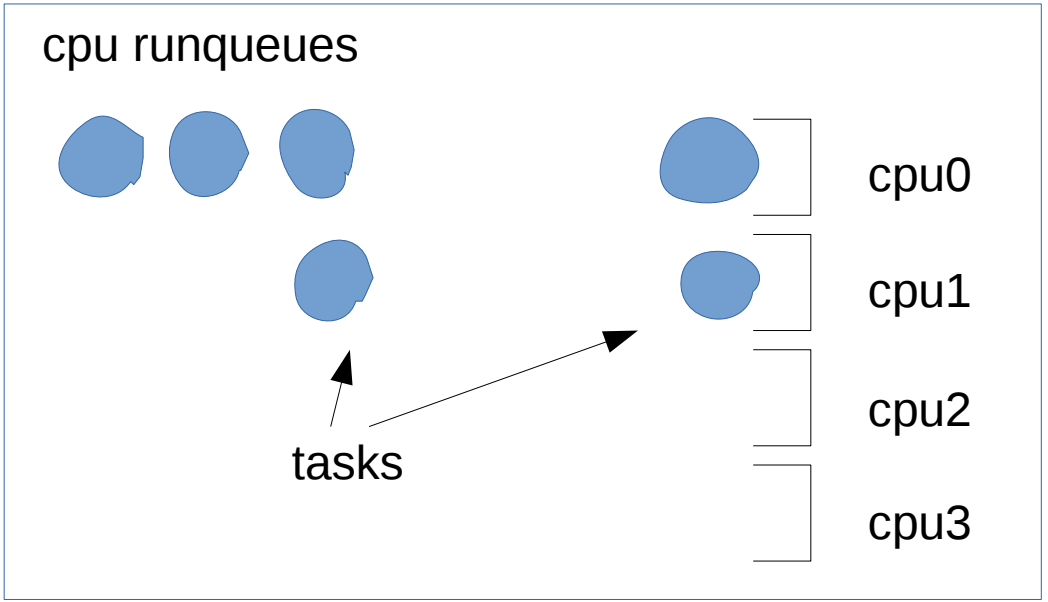
integral
considers how long error persisted

derivative
Prevents overshoot

By default, B = C = 0

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

Intermezzo: Load VS Utilization



> cpus see utilization
> scheduler knows load (runqueues, iowait)

schedutil (generic) governor (since Linux v4.7)

> input: load computed by the scheduler*

```
util_freq_invariant = util_raw * curr_freq / max_freq;

next_freq ∝ max_freq * util_freq_invariant / cpu_capacity;
```

* using the PELT algorithm https://lwn.net/Articles/531853/

kernel/sched/cpufreq_schedutil.c

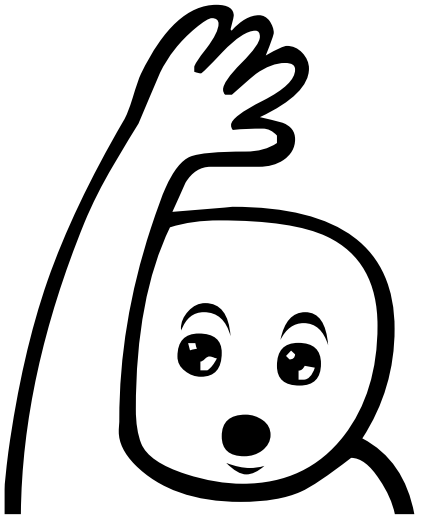
Questions!

> active power management, p-states, pairs (V, f)

> $P \propto C V^2 f + P_s$

> governors and drivers

> freq scaling w/ data from CPU VS data from scheduler



docs: https://www.kernel.org/doc/html/v4.12/admin-guide/pm/cpufreq.html

Extras

intel_pstate powersave governor

> implementation

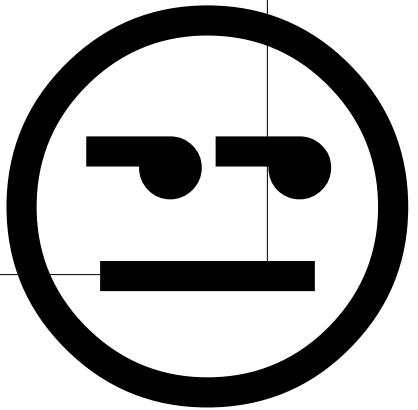
```
busy ∝ APERF / MPERF;
error = setpoint – busy;

p_term = d_gain * error;
integral += error;
i_term = i_gain * integral;
d_term = d_gain * (error – last_error);

correction = p_term + i_term + d_term;
next_pstate = current_pstate – correction;
```

drivers/cpufreq/intel_pstate.c

intel_pstate powersave governor

> default parameters

```
static struct pstate_adjust_policy pid_params = {
    .sample_rate_ms = 10,
    .sample_rate_ns = 10 * NSEC_PER_MSEC,
    .deadband = 0,
    .setpoint = 97,
    .p_gain_pct = 20,
    .d_gain_pct = 0,
    .i_gain_pct = 0,
};
```



all zero except the proportional term...

ondemand (generic) governor (since long ago)

> input: idle time

```
idle_time = cur_idle_time - prev_cpu_idle;
time_elapsed = update_time - prev_update_time;

load = 100 * (time_elapsed - idle_time) / time_elapsed;

freq_next = min_f + load * (max_f - min_f) / 100;
```

drivers/cpufreq/cpufreq_ondemand.c