# The Linux Scheduler a Decade of Wasted Cores

EuroSys’16

Cezary Siłuszyk

# Agenda

- Completely Fair Scheduler (CFS)
- Load balancing algorithm
- Bugs & fixes
- Tools
- Conclusions

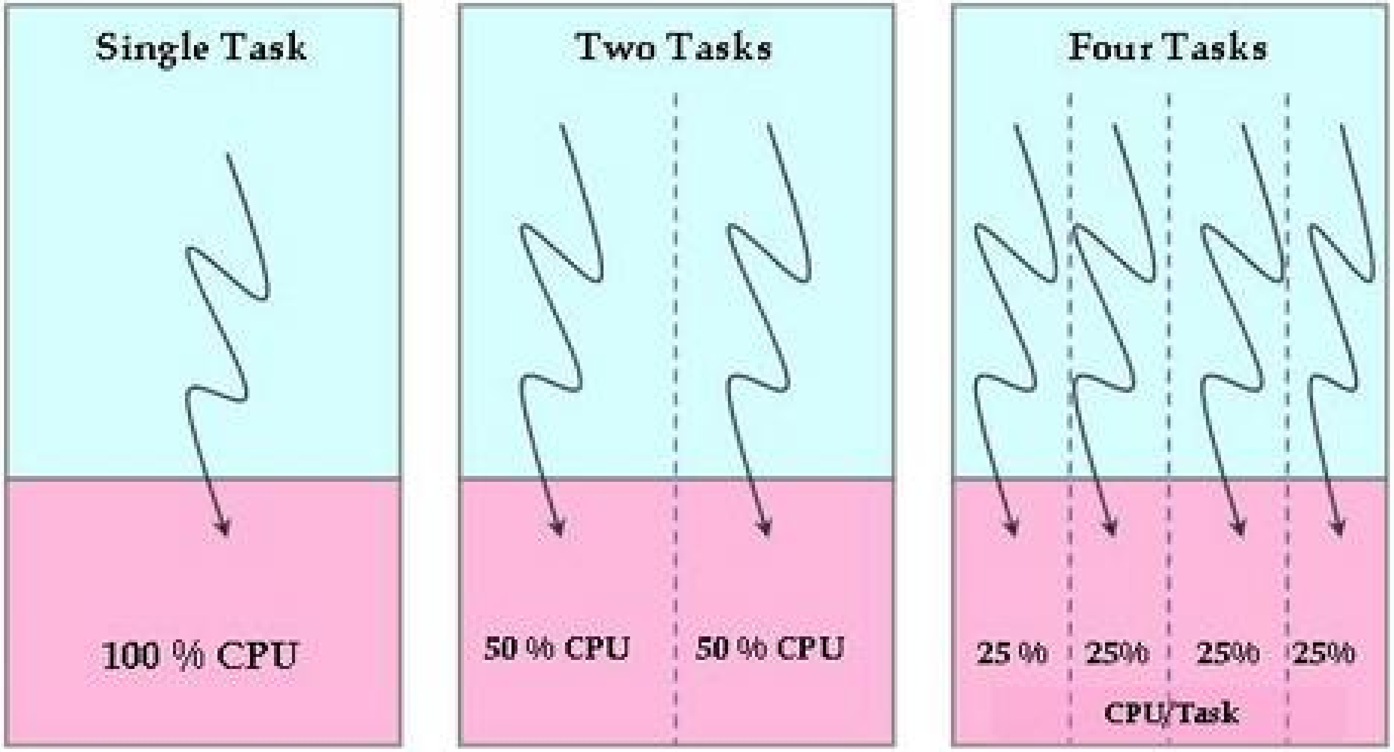
*“And you have to realize that there are not very many things that have aged as well as the scheduler. Which is just another proof that scheduling is easy.”*

Linus Torvalds, 2001

*”I wrote the first line of code of the CFS patch this week, 8am Wednesday morning, and released it to lkml 62 hours later, 22pm on Friday.”*

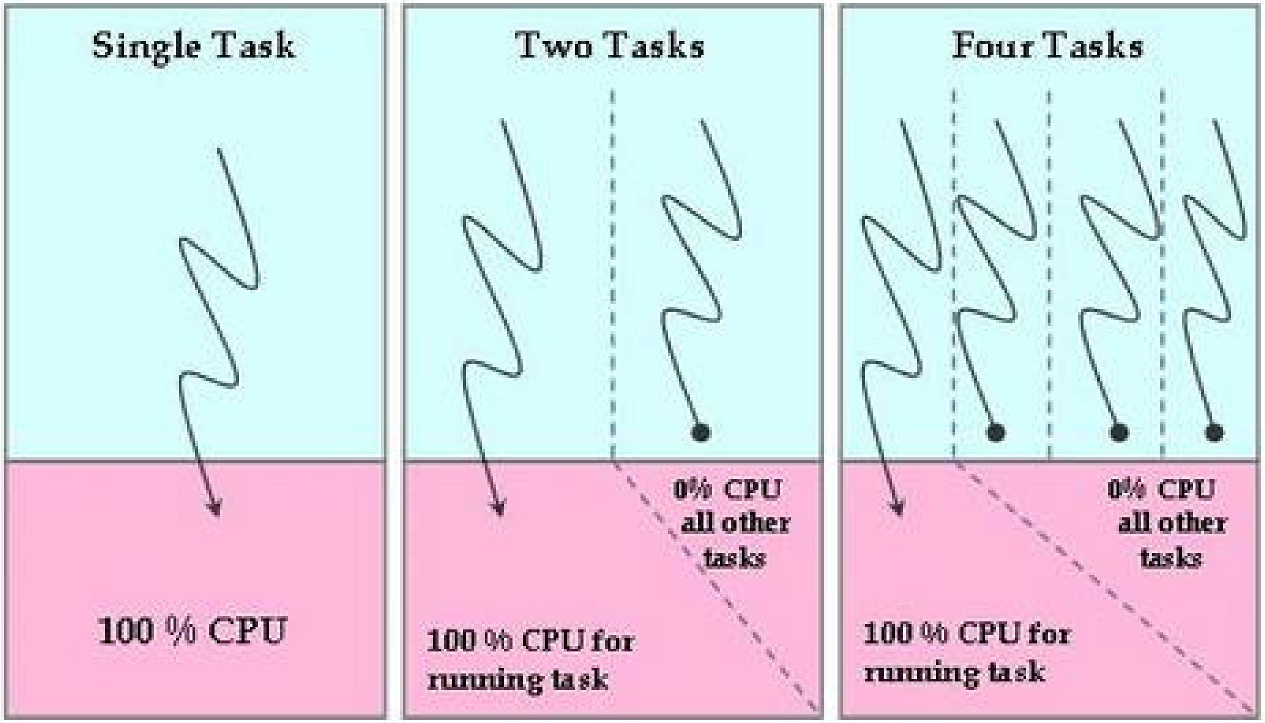
Ingo Molnar, 2007

# Ideal Precise Multi-tasking CPU



**Ideal Precise Multi-tasking CPU** – Each task runs in parallel and consumes equal CPU share

# Actual CPU



**Actual CPU** – While one task uses the CPU, every other task waits

# CFS on single-CPU system

- CFS basically models an "ideal, precise multi-tasking CPU" on real hardware
- vruntime = runtime / weight
- uses a time-ordered rbtree to build a "timeline" of future task execution – O(lg n)

# CFS on multi-core systems

- Context switch must be fast
- Core-local queues to avoid synchronization
- Requires load balancing

*“I suspect that making the scheduler use per-CPU queues together with some inter-CPU load balancing logic is probably trivial . Patches already exist, and I don’t feel that people can screw up the few hundred lines too badly.”*

Linus Torvalds, 2001

# Straightforward approach

- Load-balancing based on number of processes

- Very cheap

- High-priority threads would get same amount of CPU time as low-priority threads

# Second approach

- Load-balancing based on thread weights

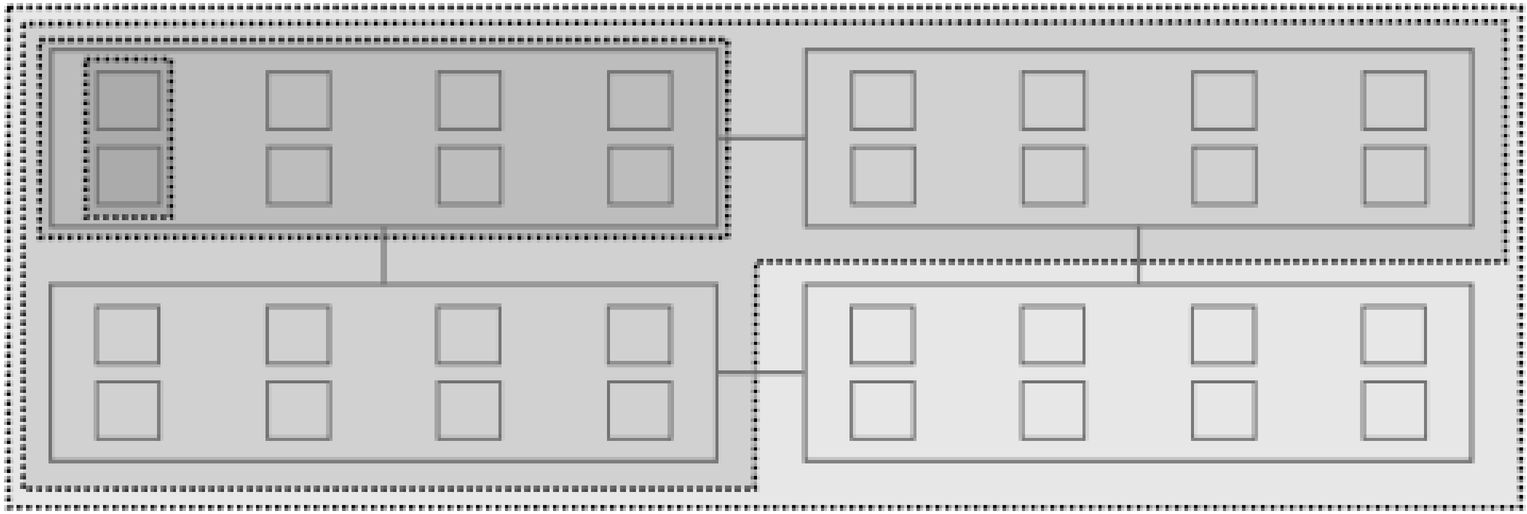
- Also cheap

- Problem: interactive high-priority threads

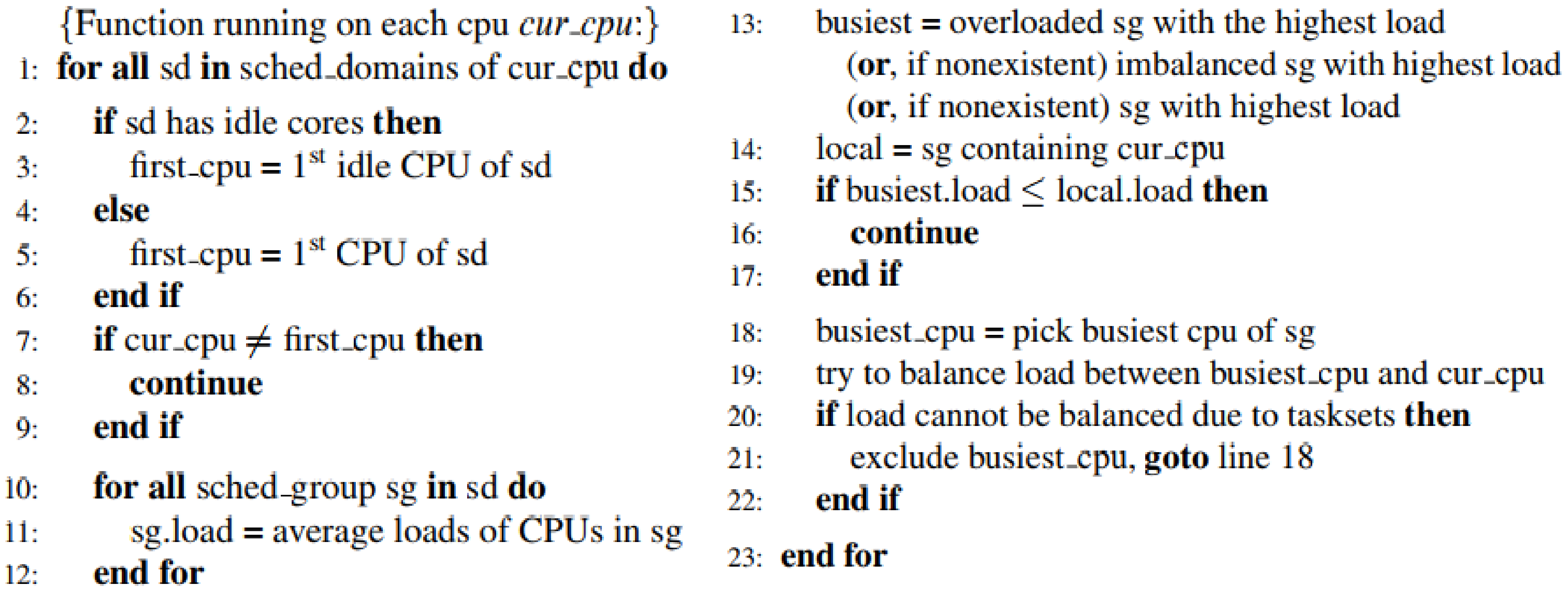
# CFS load balancing

- Load is the combination of the threads’ weight and its average CPU utilization
- cgroup feature
- Aware of cache locality

# Scheduling domains



# Load balancing algorithm



```
    {Function running on each cpu $cur\_cpu$:}
1:  for all sd in sched_domains of cur_cpu do
2:      if sd has idle cores then
3:          first_cpu = 1st idle CPU of sd
4:      else
5:          first_cpu = 1st CPU of sd
6:      end if
7:      if cur_cpu $\neq$ first_cpu then
8:          continue
9:      end if
10:     for all sched_group sg in sd do
11:         sg.load = average loads of CPUs in sg
12:     end for
13:     busiest = overloaded sg with the highest load
          (or, if nonexistent) imbalanced sg with highest load
          (or, if nonexistent) sg with highest load
14:     local = sg containing cur_cpu
15:     if busiest.load $\leq$ local.load then
16:         continue
17:     end if
18:     busiest_cpu = pick busiest cpu of sg
19:     try to balance load between busiest_cpu and cur_cpu
20:     if load cannot be balanced due to tasksets then
21:         exclude busiest_cpu, goto line 18
22:     end if
23: end for
```

*“Nobody actually creates perfect code the first time around, except me. But there’s only one of me.”*

Linus Torvalds, 2007

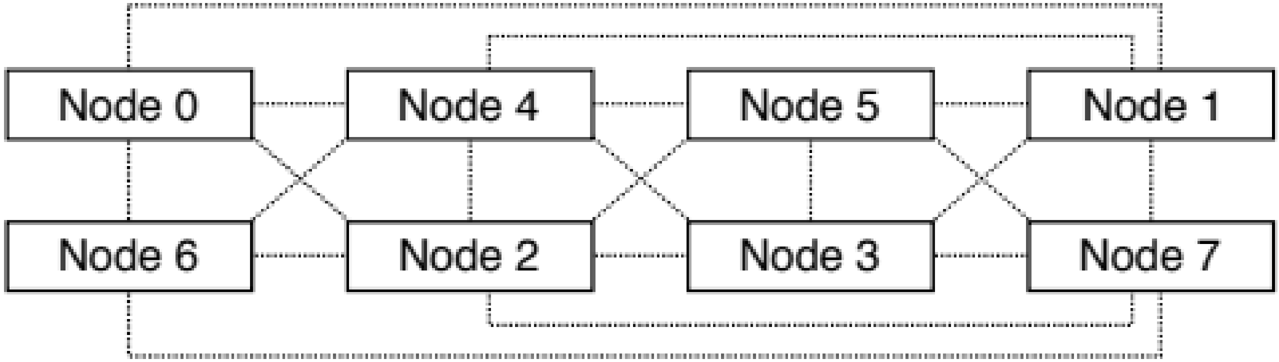
# The Group Imbalance bug

- Load balancing is based on average load

- Fix: change average to minimum

# The Scheduling Group Construction bug

taskset enables pinning applications to run o a subset of available cores. Groups are constructed from the perspective of a specific core (0), but they should be constructed from the perspective of the core responsible for load balancing on each node.



# The Overload-on-Wakeup bug

- Introduced by an optimization in the wakeup code

- Scheduler attempts to place the woken up thread physically close to the waker thread

- Fix: wake up thread on idle core

# The Missing Scheduling Domains bug

- When a core is disabled and then re-enabled using the /proc interface, load balancing between any NUMA nodes is no longer performed

- Incorrect update of a global variable representing the number of scheduling domains in the machine

*“All our fixes will be submitted to the kernel developers shortly”*

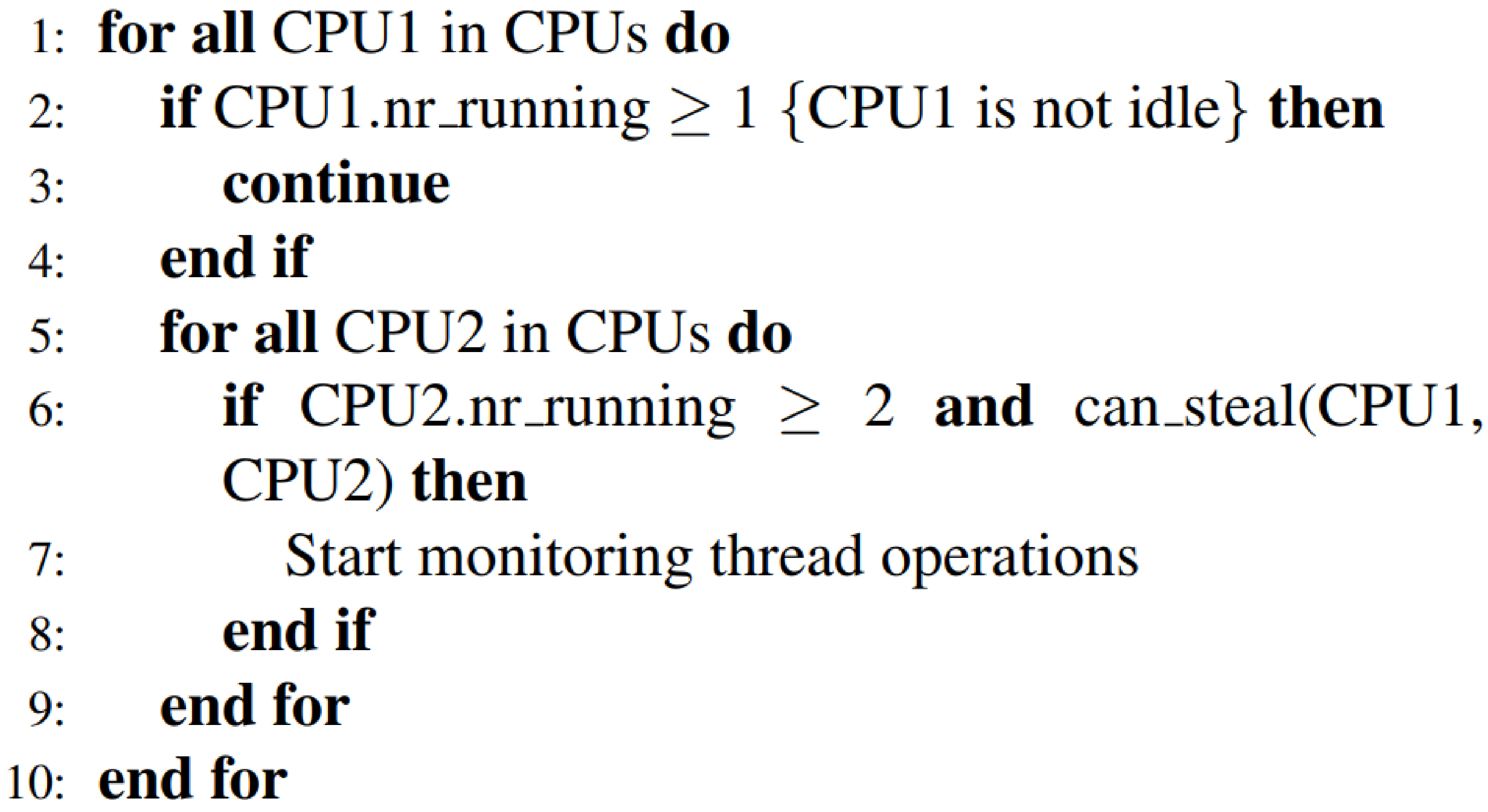
Authors

# lkml response

- “their patches are completely butchering things”, Peter Zijlstra

- “One of the issues has been fixed, one is a non-issue and we had ideas about at least one other and I cannot quite remember what the 4th was.”, Peter Zijlstra

# Online Sanity Checker



```
for all CPU1 in CPUs do
    if CPU1.nr_running $\geq$ 1 {CPU1 is not idle} then
        continue
    end if
    for all CPU2 in CPUs do
        if CPU2.nr_running $\geq$ 2 and can_steal(CPU1, CPU2) then
            Start monitoring thread operations
        end if
    end for
end for
```

# Scheduler Visualization tool

- Allows to plot
  - size of run queues
  - total load of run queues
  - cores that were considered during periodic load balancing and thread wake-ups

- Visualizations generated by sh script using PHP

# Lesson Learned

- Bugs resulted from optimizations
- Visualization is a good idea
- Fixes not merged to mainline (not even proposed)
- Catchy paper name matters

# Bibliography

- Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, Alexandra Fedorova - The Linux Scheduler: a Decade of Wasted Cores (EuroSys'16)
- doc/Documentation/scheduler/sched-design-CFS.txt
- http://www.linuxjournal.com/magazine/completely-fair-scheduler

# Questions?