What is cooking inside the Linux task scheduler?

18 January 2024

Himadri Chhaya-Shailesh
PhD Student at Whisper, Inria Paris
Supervised by Julia Lawall & Jean-Pierre Lozi

Outline

About me

Previous Work

- Supporting larger page sizes for hyper-V guests on ARM64[1]
- Fixing short reads in `usb_control_read()`[2]

[1] https://himadrics.tumblr.com/post/187125172565/modifying-expectations

[2] https://himadrics.tumblr.com/post/634481719919165440/linux-kernel-bug-fixing-mentorship

Current Research

- The Nest Scheduler (EuroSys'22)[3]
- Paravirt Scheduling for QEMU/KVM Guests

[3] https://inria.hal.science/hal-03612592/file/paper.pdf

Co-mentoring

- Semester Long Projects at DAIICT[4]
- Linux Foundation mentorship program for Linux Kernel Bug Fixing [5]

[4] https://medium.com/code-dementia/a-beginners-journey-into-linux-kernel-d315f0af4757

[5] https://blog.realogs.in/experience-of-lfx-mentorship/

History

The Very First Scheduler (Linux v0.01, 1991)

Inria

~500 LoC

```
File: include/linux/sched.h
---
#define NR_TASKS 64
#define HZ 100

File: kernel/sched.c
---
void schedule(void)
{ ...
  while (1) { ...
  for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
    if (*p)
      (*p)->counter = ((*p)->counter >> 1) +
                                  (*p)->priority;
  }
  switch_to(next);
}
```

Scheduler Evolution[6]

- 1999 - Scheduling classes (SCHED_RR and SCHED_FIFO)
- 2001 - O(n) Scheduler
- 2003 - O(1) Scheduler
- 2007 - CFS
- 2014 - SCHED_DEADLINE
- 2023 - EEVDFS

[6] Brief History of Linux CPU Scheduler - Huichun Feng, FOSSASIA'22

The Scheduler Today (Linux v6.8-rc1, 2024)

Inria

~50,000 LoC that deals with

- CPU topologies (SMP and NUMA)
- Load balancing
- CPU frequency and utilization
- Group scheduling
- Core scheduling
- Energy-aware Scheduling
- Realtime scheduling

EEVDFS

EEVDFS has been around

Inria

Earliest Eligible Virtual Deadline First Scheduler

- It is not really NEW!
- First proposed by Ion Stoica and Hussein Abdel-Wahab in 1995
- Peter Zijlstra, the Scheduler maintainer, tried to implement it earlier in 2010
- Merged into the Linux kernel in August 2023 (v6.6)

Why Now?

- Processes has no way to express their latency requirements to the Scheduler with CFS
- Latency nice patch series (2019)
- Zijlstra revisited EEVDFS idea to address the latency issue

The big idea behind EEVDF algorithm

Inria

- Proportional Share schedulers are good in achieving fairness
- Real time schedulers are good in achieving deadlines

While retaining all the advantages of a proportional share scheduler, EEVDF provide strong timeliness guarantees

What doesn't change?

- Most of the scheduler code
- The notion of vruntime
- The load balancing algorithms

What changes?

- The selection of which new task to run
- CFS selects the task with least vruntime
- EEVDFS selects the task with the earliest virtual deadline

Lag

The difference between the processor time that process should have gotten and how much it actually got is defined as the lag

- A positive lag value means that the process has not received its fair share and it should be scheduled sooner
- A negative lag value means that the process has received more than its fair share and it should be slowed down

Eligibility

A process is deemed to be "eligible" if - and only if - its calculated lag is greater than or equal to zero

- A currently ineligible process becomes eligible in the future when currently eligible processes catch up
- The time when such an ineligible process becomes eligible again is called eligible time

Virtual Deadline

The earliest time by which a process should have received its due CPU time is defined as the virtual deadline

virtual deadline = allocated time slice + eligible time

Addressing the latency problem[7]

- Latency nice values are taken into account while calculating the time slice value of a process
- A process with a lower latency-nice setting (and, thus, tighter latency requirements) will get a shorter time slice
- Processes that are relatively indifferent to latency will receive longer slices
- The amount of CPU time given to any two processes (with the same nice value) will be the same, but the low-latency process will get it in a larger number of shorter slices

[7] `https://lwn.net/Articles/925371/`

It works because...

- virtual deadline = allocated time slice + eligible time
- Latency sensitive processes normally don't need large amounts of CPU time
- With shorter time slices they will have closer virtual deadlines
- So they will be executed first and get to respond quickly to events
- The effect will be achieved without adding anymore tricky scheduler heuristics

Implementation details

- 265 new lines in `fair.c`
- `entity_eligible()`
- `__update_min_deadline()`
- `min_deadline_update()`
- `pick_eevdf()`
- `pick_cfs()`
- `set_slice()`

sched_ext

Motivation[8]

CFS works appreciably well across a large number or hardware architecture and for a variety of diverse workloads
But...

- It was built in a simpler time
- Experimentation with CFS is difficult
- It often takes O(years) for people to fully understand CFS' implementation details
- It is difficult to get new features merged upstream

[8] https://kernel-recipes.org/en/2023/schedule/sched_ext-pluggable-scheduling-in-the-linux-kernel/

eBPF

Inria

A revolutionary technology to safely run user-defined and event driven code at kernel runtime

sched_ext[10]

- Allows writing Schedulers in eBPF
- A new scheduling class - SCHED_EXT
- If your eBPF scheduler misbehaves, then fall back to the default kernel Scheduler
- Production ready example Schedulers[9]

[9] https://github.com/sched-ext/scx/tree/main/scheds
[10] https://lwn.net/Articles/922405/

Nest

Per-core task scheduling in Linux

Inria

The goal of a task scheduler

- Place tasks on cores on fork, wakeup, or load balancing
- Choose a task on the core to run when the core becomes idle

The challenge

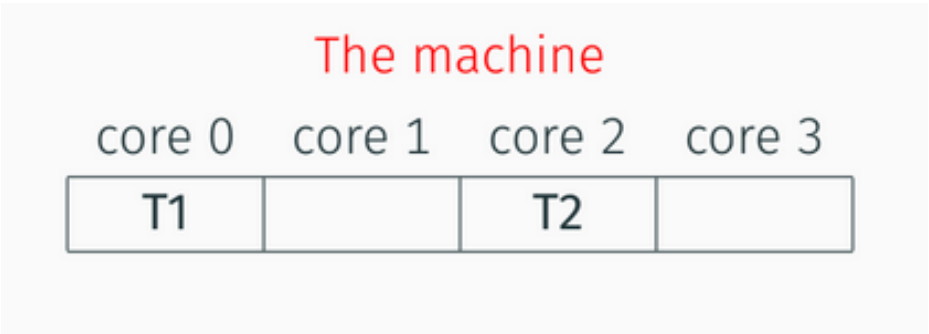
- Task placement that synergizes with hardware features

[Slide by Julia Lawall]

Work conservation

If a core is overloaded, no other core should be idle

Example



Where to put waking thread T3?

[Slide by Julia Lawall]

Dynamic Voltage and Frequency Scaling

Inria

DVFS - A hardware feature that enables cores to run at different frequencies

Higher frequency ->

- faster execution
- more energy consuption
- more heat generation

- More activity on a core results in a higher frequency
- Only few cores of the machine are allowed to run at the highest/turbo frequencies at any given time

[Slide by Julia Lawall]

Impact of DVFS on scheduling

Where to put waking thread T3?

	core 0	core 1	core 2	core 3
recent	T1	T0	T2	
current	T1		T2	

Core 1 could be a better choice

- Core 1 was recently active, so at a higher frequency
- Core 3 would suggest there are 4 active cores, giving a lower turbo frequency

[Slide by Julia Lawall]

The big idea behind Nest algorithm

Inria

Exploit core frequencies while making task placement decisions

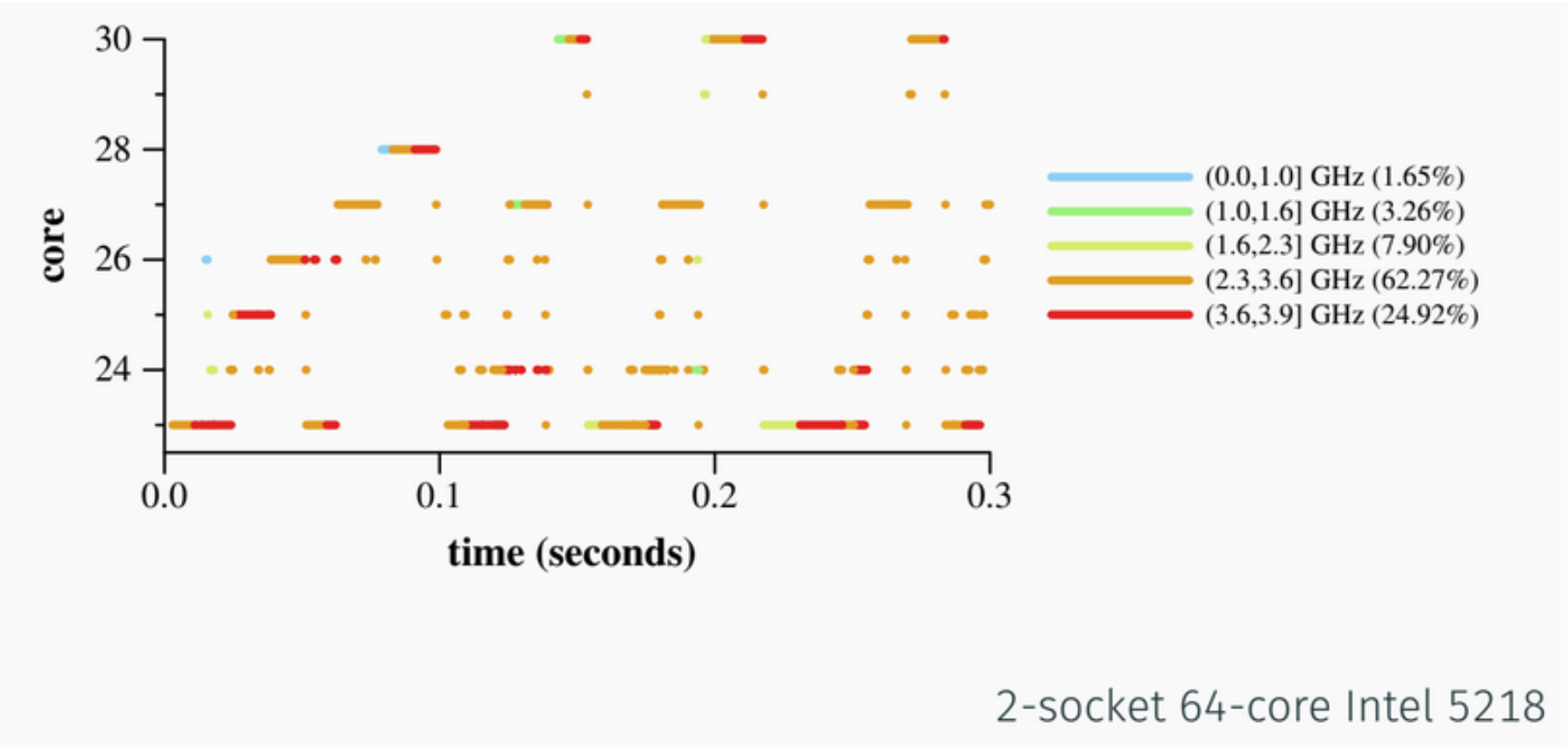
Reuse cores

- Maintain a nest of recently used cores to acheive core

Keep cores warm

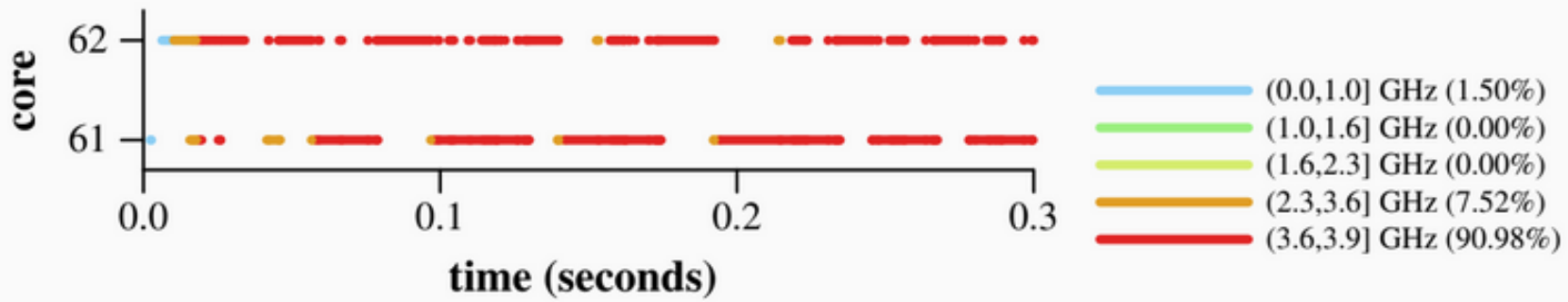
- Cores in the nest are likely to be reused, so spin briefly when they go idle, to keep the frequency high.

Task Placement with CFS



[Slide by Julia Lawall]

Task Placement with Nest



2-socket 64-core Intel 5218
16% speedup overall

[Slide by Julia Lawall]

Implementation details[11]

- Primary nest
- Reserve nest
- Nest compaction
- Impatient tasks
- Idle process spinning

[11] The eBPF implementation of Nest is now available as an example with sched_ext

Schedgraph

Schedgraph[12]

Graphing tools for Scheduler developed by Julia Lawall

- `dat2graph` - Horizontal bar graph showing what is happening on each core at each time
- `running_waiting` - Line graph of how many tasks are running or waiting on a runqueue at any point in time
- `stepper` - Step-by-step execution of all tasks on all cores
- `hostguest` - Activity on vcpus + status of vcpus as running or waiting

[Slide by Julia Lawall]

[12] `https://gitlab.inria.fr/schedgraph/schedgraph`

Semantic Gap

Context - Virtualization

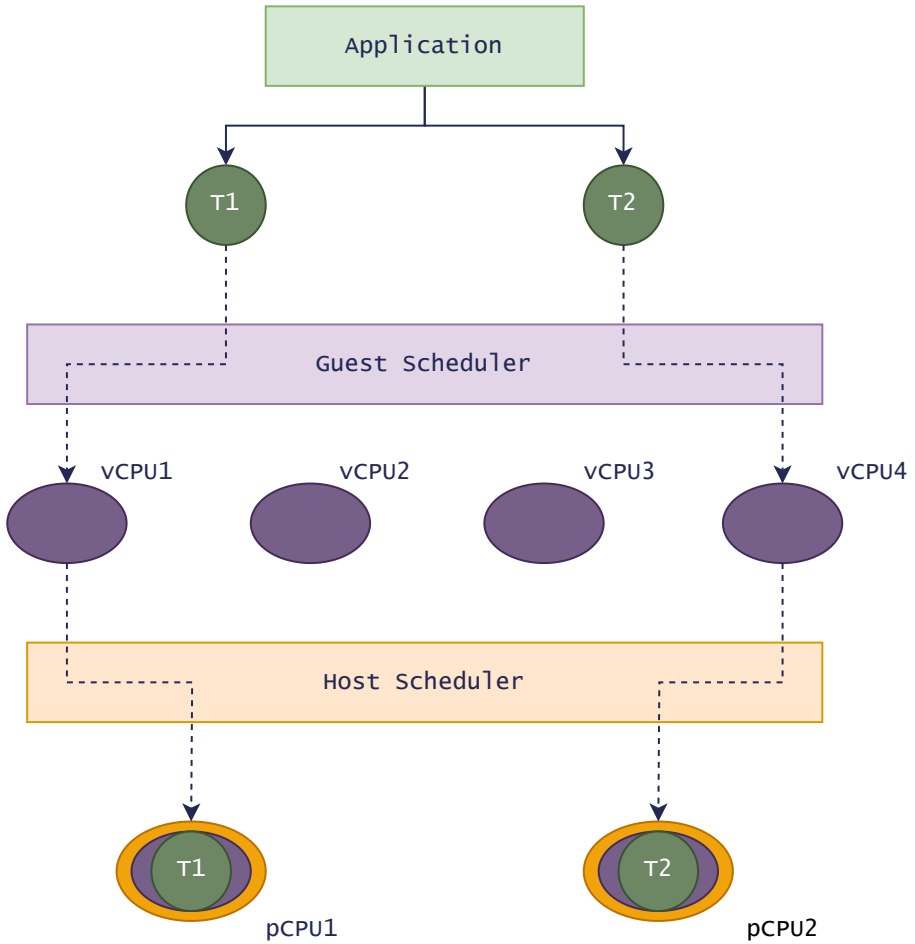


Figure 1: Dual level of task scheduling for VM workloads

What makes this context an interesting research problem?

The semantic gap between the schedulers:

- Neither the guest scheduler nor the host scheduler has all the necessary information to make correct task placement decisions
- The guest and the host schedulers do not share the information about their task placement decisions with each other

What makes this context an interesting research problem?

Missing information at the guest level:

- Changes in vNUMA topology at the runtime[13]
- State of the vCPUs (running/ waiting/ sleeping) on the host[14]

[13] Bao Bui at el. address this issue in "When EXtended Para - Virtualization (XPV) Meets NUMA" [EuroSys '19]

[14] `vpu_is_preempted()` interface in the Linux kernel partially addresses this issue

What makes this context an interesting research problem?

Missing information at the host level:

- Threads' activities inside the guest[15]

[15] It leads to the well-known lock holder preemption problem

vCPU States

We define:

- a vCPU that is waiting in a runqueue of a pCPU on the host as a **Phantom vCPU**
- a vCPU that is either running on a pCPU or sleeping as a **Viable vCPU**

Phantom vCPU Example

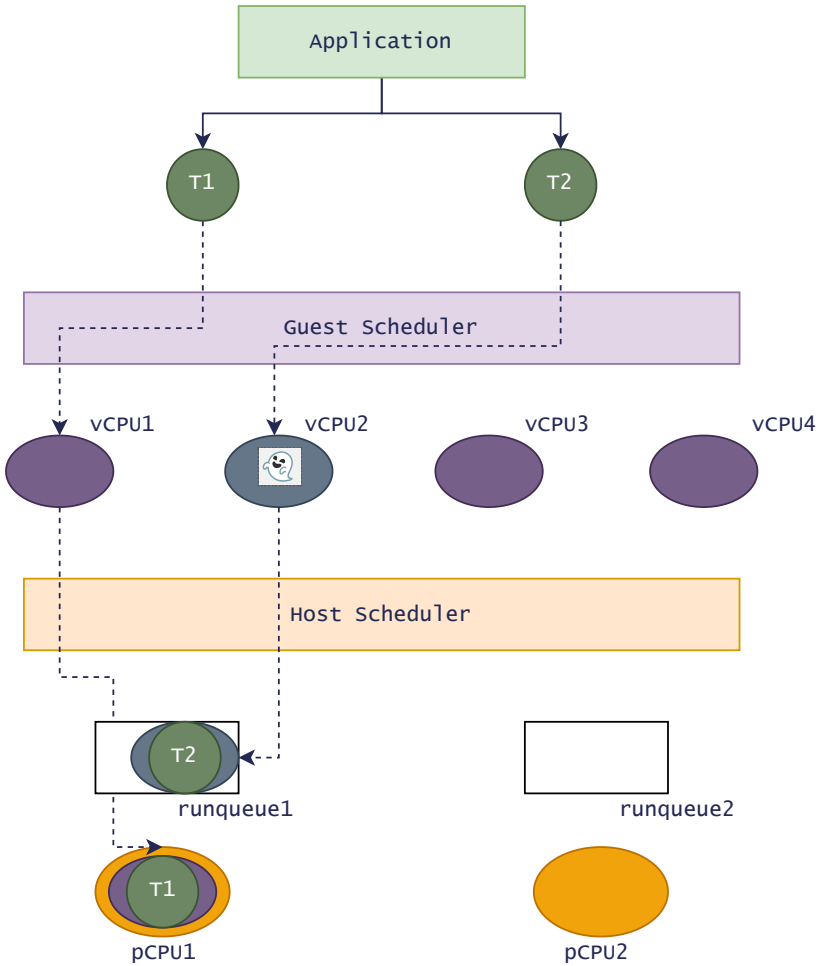


Figure 2: Phantom vCPU in an oversubscribed guest

How does a vCPU become a Phantom?

A vCPU becomes a phantom in two scenarios:

1. When all the pCPUs are busy, and the vCPU is unable to preempt the task running on the busy pCPU[16]
2. An idle pCPU is available, but the vCPU still ends up in the run queue of a busy pCPU[17]

[16] It is a common occurrence in oversubscribed and multi tenant hosts
[17] It occurs when CFS fails to ensure work conservation

Existing measures against phantom vCPUs

- The guest scheduler uses `vcpu_is_preempted()` interface on `x86` architecture, to avoid preempted vCPUs while searching for an idle vCPU to place a task [18]

- KVM implements the accounting of the `steal_time` for vCPUs and does the correction in `vruntime` to ensure that the task is not penalized when the vCPU is preempted

[18] It is limited to the search of an idle core, and hence a preempted vCPU can still get selected if no idle cores are available. Moreover, the interface is not implemented for other architectures yet

Para-virt Scheduling

- A communication interface between the host and the guest schedulers
- Share scheduling information between the two schedulers
- Avoid preempting vCPUs running sensitive tasks on the host
- Enable host to directly schedule the tasks running inside the guest
- And more?

Take away

- EEVDFS - the new scheduler in the town
- sched_ext - the eBPF way of writing Schedulers
- Nest - a scheduler that targets DVFS hardware
- Schedgraph - graphing tools for Scheduler
- Paravirt Scheduling - addresses the semantic host and the guest