Sched Ext

What's New (& What's Missing)?

David Vernet
Kernel engineer

Meta

Agenda

01 Background & current upstream status

02 Latest and greatest features

03 What do we still need?

01 Background & current upstream status

Refresher on sched_ext

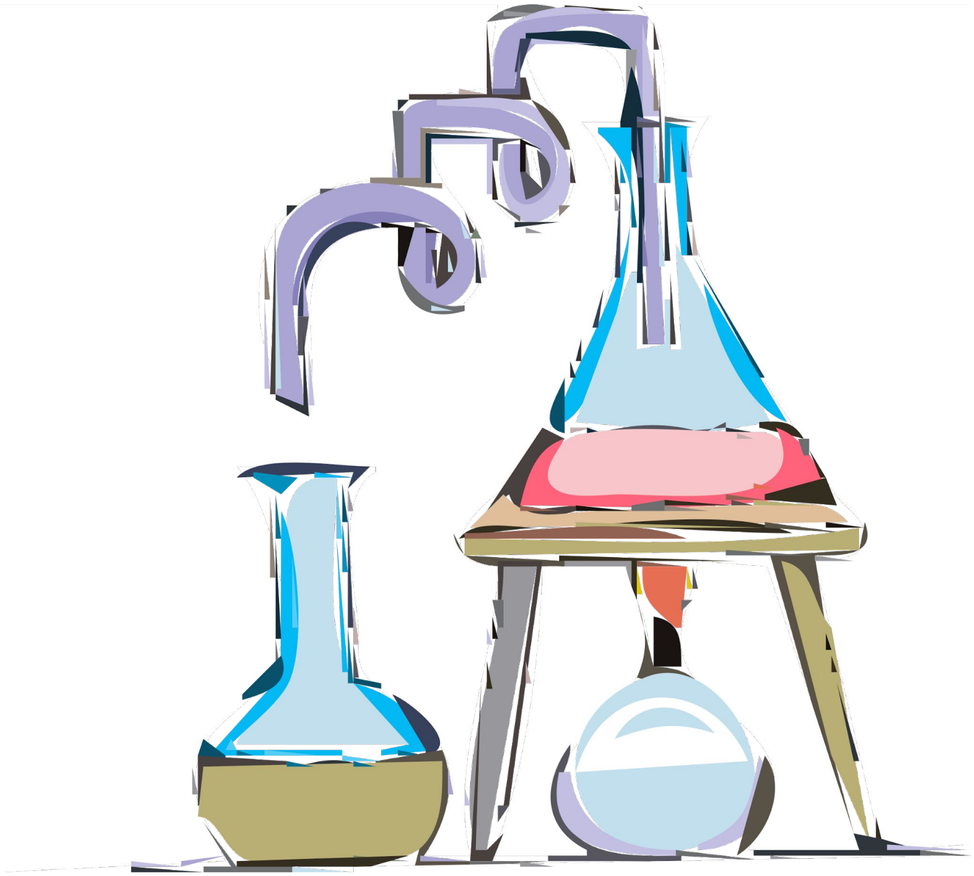
sched_ext enables scheduling policies to be implemented in BPF programs

1. Write a scheduler policy in BPF
2. Compile it
3. Load it onto the system, letting BPF and core sched_ext infrastructure do all of the heavy lifting to enable it

- New sched_class, at a lower priority than CFS
- No ABI stability restrictions – purely a kernel <-> kernel interface
- GPLv2 only

Rapid experimentation

- No reboot needed – just recompile BPF prog and reload
- Simple and intuitive API for scheduling policies
  - Does not require knowledge of core scheduler internals
- Safe, cannot crash the host
  - Protection afforded by BPF verifier
  - Watchdog boots sched_ext scheduler if a runnable task isn't scheduled within some timeout
  - New sysrq key for booting sched_ext scheduler through console
- See what works, then implement features in EEVDF



Bespoke scheduling policies

- EEVDF is a general purpose scheduler. Works OK for most applications, not optimal for many
- Optimizes some major Meta services (more on this later)
  - **HHVM optimized by 1-3+% RPS**
  - **Looking like a 3.6 - 5+% improvement for ads ranking**
- Google has seen strong results on search, VM scheduling with ghOSt
- Valve + Igalia seeing strong results on p99 frame rate latency on the Steam Deck

Moving complexity into user space

- Offload complicated logic such as load balancing to user space
- Avoids workarounds like custom threading implementations and other flavors of kernel bypass
- Use of floating point numbers
- BPF makes it easy to share data between the kernel and user space

v6 was sent in early May

- https://lore.kernel.org/bpf/20240501151312.635565-1-tj@kernel.org/
- Discussion is still contentious, but lots to be excited about

The community has grown by a **lot**

- Valve, in cooperation with Igalia, are planning to ship a sched_ext scheduler on Steam Deck
- Ubuntu is considering shipping sched_ext in the official, upcoming 24.10 release
  - Andrea Righi is actively developing a user space rust scheduler called scx_rustland
- Meta is deploying a sched_ext scheduler for its web workload, and is in the process of rolling out for ads and ML training
- Google planning to port ghOSt to sched_ext down the line
- ChromeOS looking into using scx_layered with a focus on reducing latency
- Oculus is experimenting with sched_ext on an Android port

02 Latest and greatest features

Since last year, a lot of improvements have landed in the core framework

- Much better debugging
  - struct_ops hooks for dumping debug information
  - Iterating over runqueues and printing out runnable task information
  - See https://drive.google.com/file/d/1G48x_E3h-WGgx1AHszKS4efV2_Hpxsy5/view?usp=drive_link for demo showing debug output
- Cpufreq integration
  - Integrates with schedutil – generic API for scaling frequency on individual cores
  - Pieces in place for implementing PELT (Per Entity Load Tracking) when needed
- Better semantics for dispatching
- New callback points such as task tick
- Better hotplug support
- Better local kptr support (**BPF**)
- Better backwards compatibility APIs for BPF progs and user space (**BPF**)

...and in the schedulers themselves

https://github.com/sched-ext/scx/tree/main/scheds/rust/scx_rusty

- **scx_rusty**: Work conserving, interactive, hybrid user space rust / BPF scheduler that implements load balancing + statistics in user space, and hot-path scheduling decisions in the kernel
  - Interactive, now seems to beat EEVDF for gaming workloads + workloads with lots of background CPU
  - Work conserving – will always give idle CPUs to tasks (unless configured otherwise)
  - NUMA aware
  - Experimenting internally at Meta

https://github.com/sched-ext/scx/tree/main/scheds/rust/scx_layered

- **scx_layered**: JSON-config driven scheduler that can be used to tune for individual workloads
  - Match tasks into “layers” by various qualifiers: task name, process name, niceness level, cgroup name
  - Specify behavior for each layer: how much CPU may be allocated, should the layer preempt other layers, etc
  - Running large portions of Meta web workloads, rolling out to others soon

rusty (borrowing concepts from lavd) fundamentally changes dl scheduling

- EEVDF decides deadline based on virtual runtime eligibility + slice length:
  - v_dl = v_ei + (slice / w_i)
  - Problem is that a task can't know how much CPU it needs *before* it runs
  - Slice can be configured, but configuring it is brittle and very difficult to get right
- rusty instead uses 3 criteria to determine deadline:
  - Task **block frequency**: Correlates with task's role as a **consumer**
  - Task **waking frequency**: Correlates with task's role as a **producer**
  - Task average runtime: Correlates with task's role as a latency sensitive task
  - **Note**: Having both high block *and* high waking frequency implies task in the middle of a task chain
  - ^ Credit to Changwoo Min for this insight
- scx_rusty calculates task deadline as:
  - v_dl = v_i + (wake_freq^2 * block_freq^3) / w_i + avg_runtime / w_i
  - Producer + consumer tasks, and tasks with shorter avg runtime, get CPU more quickly
  - Because deadline is based off of task vruntime, fairness is not violated

Result: *significantly* better interactivity

- Throw huge amounts of load at the system, can still play games, listen to Spotify, etc
  - Or for servers, can still service latency-sensitive tasks
- scx_rusty is work conserving, so typically still beats EEVDF on throughput as well
  - See https://github.com/sched-ext/scx/pull/261 for more details
  - See https://drive.google.com/file/d/1fyHt9BYGha6apI7HAkibwpy52UTi8-AQ/view for demo video showing wins

...and in the schedulers themselves cont.

https://github.com/sched-ext/scx/tree/main/scheds/rust/scx_lavd

- **scx_lavd**: Interactive, gaming-driven scheduler optimized for Steam Deck
  - Built by Changwoo Min at Igalia, designed to run on Steam Deck
    - Currently, Valve planning to ship this scheduler in a future Steam Deck release
  - Highly novel, effective techniques for interactivity, preemption
  - Not (yet) NUMA or CCX aware

https://github.com/sched-ext/scx/tree/main/scheds/rust/scx_rustland

- **scx_rustland**: User space rust scheduler, optimized for interactivity
  - Built by Andrea Righi at Canonical
  - Most logic in user space in rust
    - Minimal logic in kernel space, mostly just applying user space decisions
  - Minimal overhead thanks to use of user -> kernel and kernel -> user ringbuffers

Schedulers located in separate github repo

- https://github.com/sched-ext/scx
- Low barrier to entry
  - Submit your scheduler, iterate on it as you want
  - Ideas can be shared and applied to other schedulers
  - Get code reviews (if you want them)
- Provides libraries and crates for working with schedulers
- Decouples packaging schedulers from the kernel release process

03 What do we still need?

Meta

First and foremost: moar BPF

- A lot of things are still challenging to do in BPF
  - Can’t hold spinlock around function calls (thanks to Kumar for improving this for static subprogs)
  - Edge cases that we end up hitting a lot
    - Can’t have a kptr inside of a nested struct
    - Can’t have array of statically allocated kptrs
    - Can’t return from bounded iterators
  - Max stack size exceeded
- Allocating and moving between stacks for certain BPF programs
  - Will be necessary to implement hierarchical cgroup scheduler (discussed more later)
- Reading + writing thread + cgroup local storage from users space
- libbpf-rs can sometimes be tricky to get right due to e.g. all of the references passed around in skeletons
- Rust bindgen – auto-create backwards-compat safe bindings?

Also needed: more people working on schedulers

- A lot of the features we need are already in place in the core kernel part of sched_ext
- Lots of people willing to run benchmarks and test schedulers (thank you!), but people working on them is less common
  - Patches are very, very welcome; to any existing schedulers
  - Feel free to write and submit your own scheduler, see few slides earlier
- What's most helpful at this point:
  - **Improving existing schedulers**, or adding new ones
  - Implementing libraries or crates that schedulers can use
  - Per Entity Load Tracking (PELT)
  - Adding tests + testing framework

Also more core sched_ext features

- Hierarchical cgroup scheduler → allow nested scheduler of schedulers for having multiple schedulers running on the host
  - Enables running in process, collaborate with user space runtime frameworks
- Other scheduler-relevant integration with rest of kernel
  - Uncore frequency tuning
    - Can become bottleneck when turbo is reached
  - Idle policy
    - Spin, enter specific cstate, etc?
    - Should maybe be entirely separate from scheduler (idle vs. sched policies are currently decoupled)
- Enabling new workloads like paravirt scheduling (shout out to Joel)
  - Will likely require work to integrate BPF with KVM to e.g. open opaque communication channels between host BPF prog <-> guest BPF prog

Joining upstream conversation

- https://lore.kernel.org/lkml/20240501151312.635565-40-tj@kernel.org/
  - The more people chiming in (especially in representation of their employer / organization), the better
  - Users will get this upstreamed


Meta