Optimizing the Linux Scheduler for Gaming … and the Road Ahead

Changwoo Min
changwoo@igalia.com

April 6th, 2026

igalia

What is the (task) scheduler?

- **Scheduler essentially decides three things:**

  (1) Which task should run first? ⇒ Deadline

  - If latency-critical tasks are delayed, it could cause stuttering.

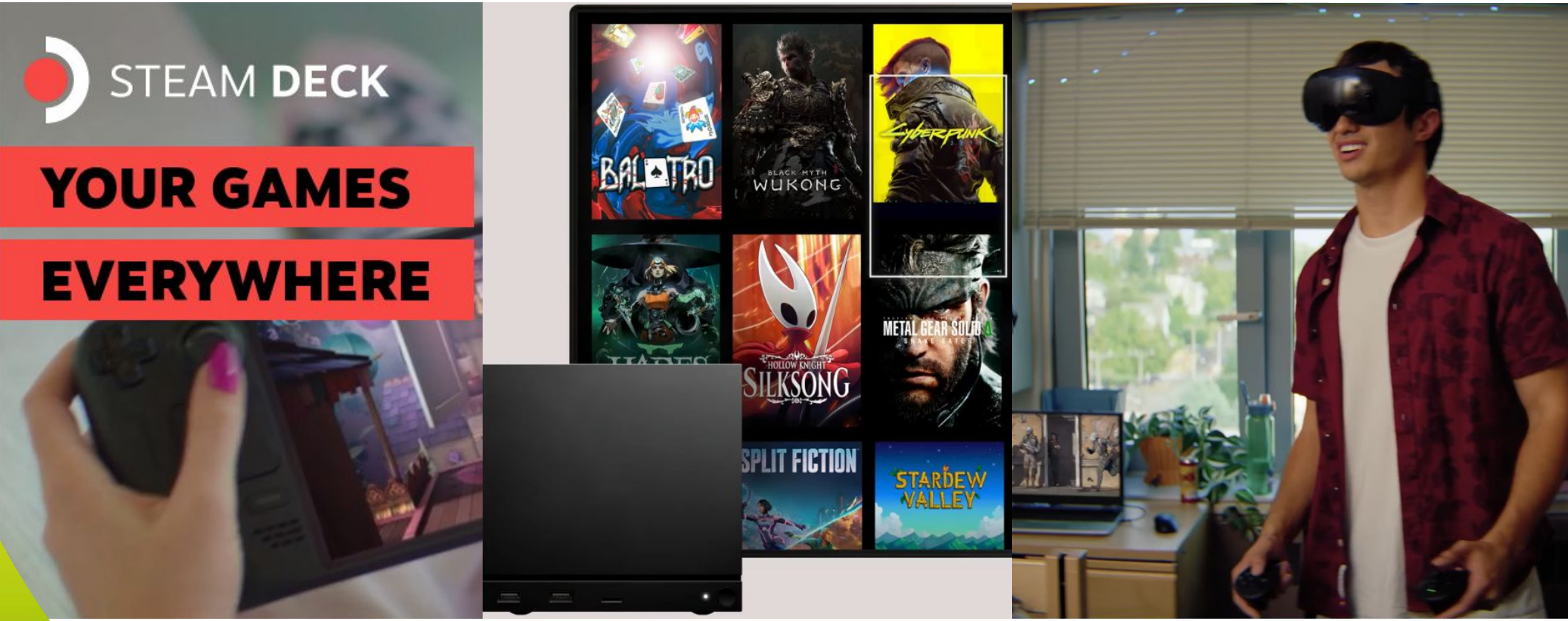
  (2) How long should the task run? ⇒ Time slice

  - A task should run long enough for warm cache, but it should not monopolize a CPU.

  (3) Which CPU should the task run on? ⇒ CPU selection

  - Choose energy-efficient core when appropriate.

Gaming on Linux is here to stay



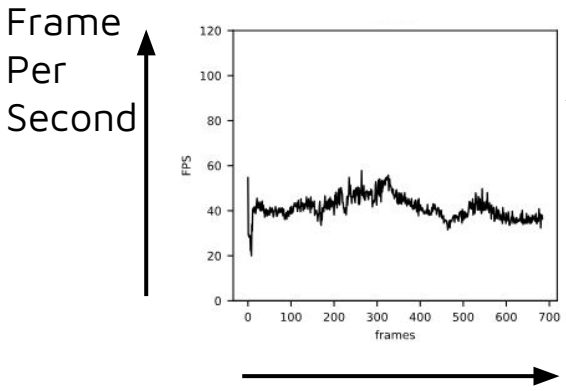
Gaming on Linux is here to stay

SteamOS runs x86 Windows games on Linux!

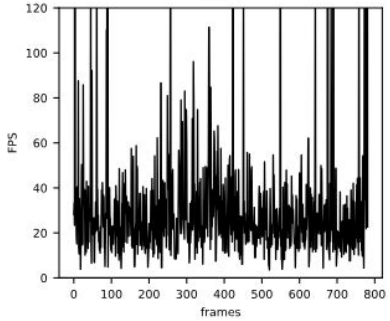
- **SteamOS runs in three different form factors:**
  - SteamDeck, Lenovo Legion Go: handheld gaming console
    - AMD x86 CPU, battery-powered
  - Steam Machine: small form factor PC for gaming
    - AMD x86 CPU, AC-powered
  - Steam Frame: VR glasses for gaming
    - Snapdragon ARM CPU, battery-powered

What is important to enhance gaming?

(1) Performance



Avg FPS: 41.3
Low 1% FPS: 29.5



Avg FPS: 34.2
Low 1% FPS: 4.6

Severe stuttering! @_@

Average FPS matters, but Low 1% FPS is also critical.

My game is stuttering even after upgrading my PC :-(

r/AMDHelp • 6 mo. ago

Games keep on stuttering, I just upgraded my pc and I can't figure out why

I recently upgraded my

MB- Gigabyte B650 Aor

CPU- Ryzen 7 7800x3d

GPU- XFX RX 7900 XF Sp

PSU- EVGA 750w Bronze

RAM- 2x16gb Corsair Ve

17 Sep, 2023 @ 12:30am

My games are stuttering

Hello guys, I bought a new pc and I started to play a few games. The ga
played were Team Fortress 2 and Assassins creed Odyssey. I mentioned
few minutes the game starts stuttering and the fps get down which dest
have with those games. My hardware consists of AMD 7, GeForce RTX 3
mainboard is B550Mx/e pro and I run the operating system windows 10.
installed the driver for my GPU and it automatically updates drivers. I als
BIOS update to fix this problem but nothing changed.
Anyone an idea what problem should get fixxed?

C. ACM, 2013

DOI:10.1145/2408776.2408794

Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUIZ ANDRÉ BARROSO

The Tail at Scale

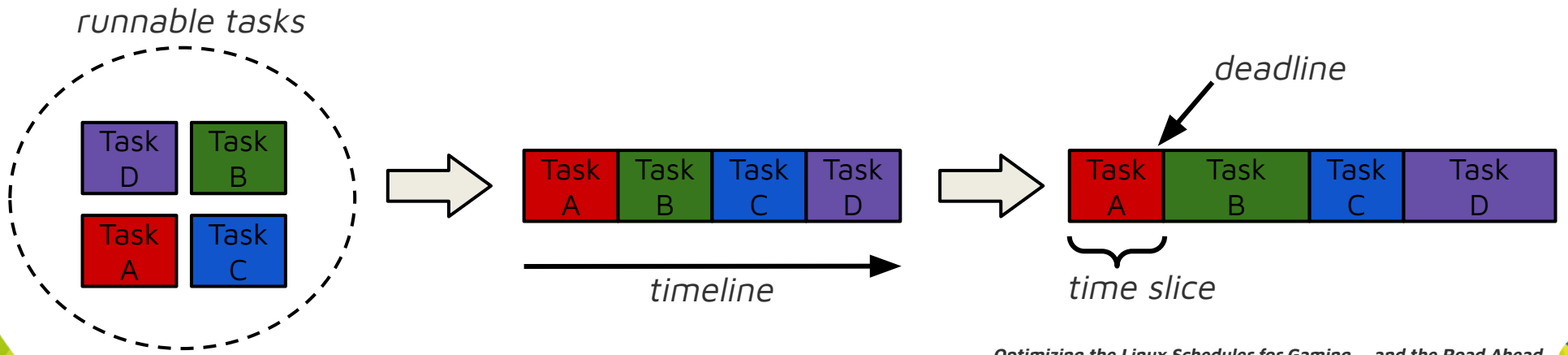
What is important to enhance gaming?

(2) Energy consumption

- Many gaming devices (e.g., SteamDeck, Steam Frame) are battery-powered.
  - So, less energy consumed, longer we can play. \^^/
- Some devices adopt hybrid processors (e.g., Qualcomm Snapdragon).
  - little core: energy-efficient but slow
  - medium core: all rounder
  - big core: boosted performance

Does scheduler matter for performance and energy consumption?

- **Scheduler essentially decides three things:**
  (1) Given a set of runnable tasks, which task should run first?
  (2) How long should each task run?
  (3) Where should the task run?



Does scheduler matter for performance and energy consumption?

(1) Given a set of tasks, which task should run first, second, etc?

- Latency-critical tasks need to be scheduled earlier to avoid stuttering.

(2) How long should each task run?

- A task should run long enough for warmer cache, etc.
- But it should not monopolize CPU.

(3) Where should the task run?

- Not all tasks are performance hungry.
- Some are OK to run on power-efficient little cores.

Any chance for gaming-optimized scheduler?

- **There are two aspects developing gaming-optimized scheduler:**

(1) Workload aspect

- If there are something unique in gaming,
  a scheduler can leverage such traits to make better scheduling decisions.

(2) Hardware aspect

- Hybrid processors are popular now. So a scheduler should make better use of hybrid processors.

Intel

CPU Specifications

Total Cores	12
# of Performance-cores	2
# of Efficient-cores	8
# of Low Power Efficient-cores	2
Total Threads	14

AMD

Series	Ryzen 7000 Series
Form Factor	Laptops, Desktops
AMD PRO Technologies	No
Regional Availability	Global , China , NA , EMEA , APJ , LATAM
Former Codename	Phoenix
Processor Architecture	2x Zen 4 , 4x Zen 4c

Qualcomm

CPU

Qualcomm® Kryo™ CPU

- 64-bit Architecture
- 1 Prime core, up to 3.4 GHz[2]
- 5 Performance cores, up to 3.2 GHz
- 2 Efficiency cores, up to 2.3 GHz

Understanding gaming workloads

- **It is critical to understand the characteristics of gaming to design optimized scheduling policies for gaming.**
- **We collected and analyzed scheduling activities while playing games.**
  - VaporMark: https://github.com/Igalia/vapormark
- **In short, gaming workloads are:**

(1) communication-intensive
  - So multiple tasks collaborate each other to accomplish a single high-level job.
  - For example, updating a game character upon a keypress event.

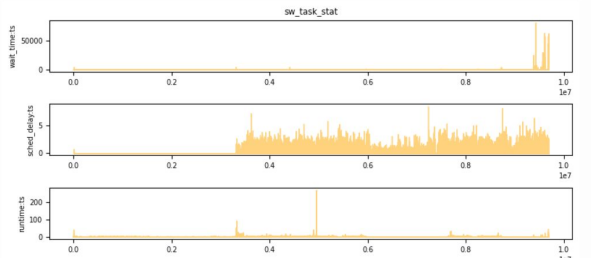
(2) short-running
  - Most tasks run for very short duration (< 1 msec).
  - They tend to yield a CPU waiting for another event.

(3) semi-periodic
  - A task does the similar work again and again.
  - It is feasible to predict a task's behavior based on its past.

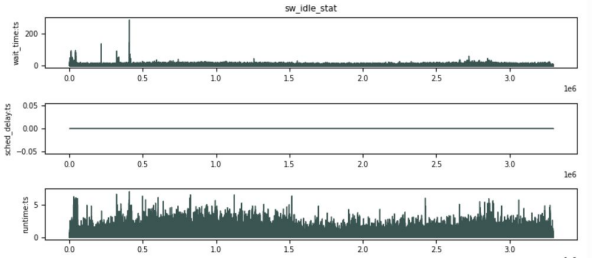
Analyzed a lot of scheduling traces

System-wide task and idle statistics

- System-wide task stats data

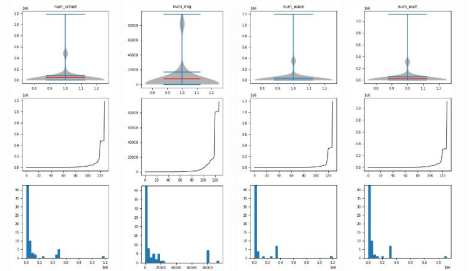


- System-wide idle stats data



Task statistics using per-task average

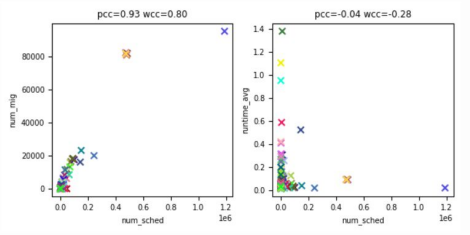
- task stats data



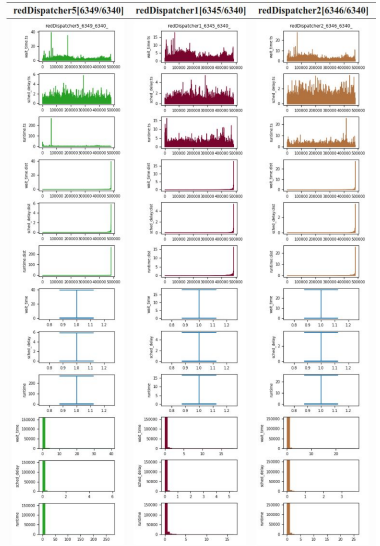
Correlation between task features

- PCC: Pearson correlation coefficient
- WCC: Weighted correlation coefficient using num_sched

- How userful is num_sched in predicting task's behavior?

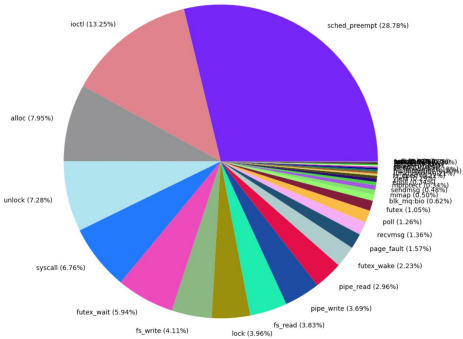


Task's wait_time, sched_delay, and runtimes

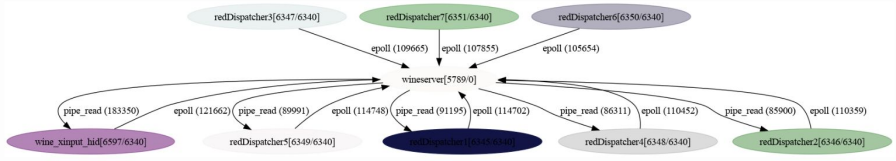


Callstacks tiggered scheduling

- callstacks
- callstack_types



- 30.00 percentile of waker-waiter

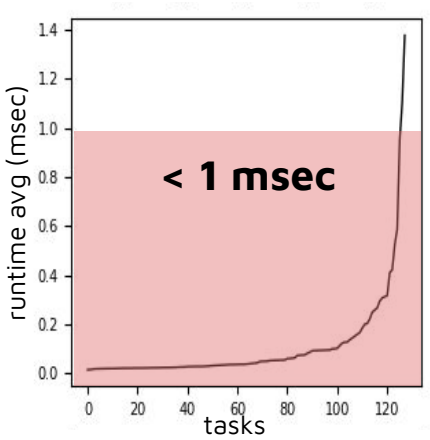


Task scheduling and CPU utilization

- **Around 300 tasks** are scheduled while running a game.
  - **Around 90% are long-living tasks**; only 10% of tasks are terminated.
- **Top 30-40** most frequently scheduled tasks take **95% of scheduling**.
  - **Around half of them** are **system tasks** -- especially, `wine`, `graphics`, and `audio servers`, taking **30--40% of scheduling**.
  - There are **15-20 game-specific tasks, which takes 60-70% scheduling.**
- **CPU utilization is moderately high -- around 65-95%**, but not overloaded (i.e., no 100%).

Task execution time per schedule

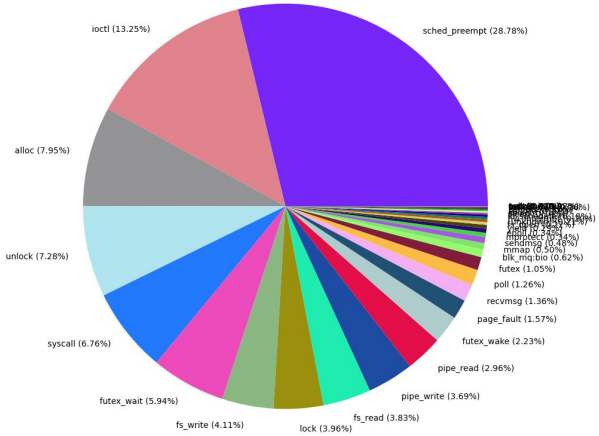
- In general, **tasks run for very short duration** – roughly **a few 100s usec** on average to **a few msec maximum**.
- **Task execution time** is very stable and is **predictable using its average.**
  - Some coordination tasks run very shortly (e.g., wineserver:260 usec) but some work tasks (e.g., a task worker: 1.65 msec) run longer than average.



Distribution of task's runtime average per schedule in a game

What makes scheduling happen?

- **Preemption (e.g., timer interrupt) takes only 25-30% of scheduling.**
- **70-75% of scheduling is initiated by waiting system calls** – such as `epoll`, `pipe_read`, `futex_wait`, etc.
- In this case, a task is first scheduled out waiting for an event then it is scheduled in when the event is notified.

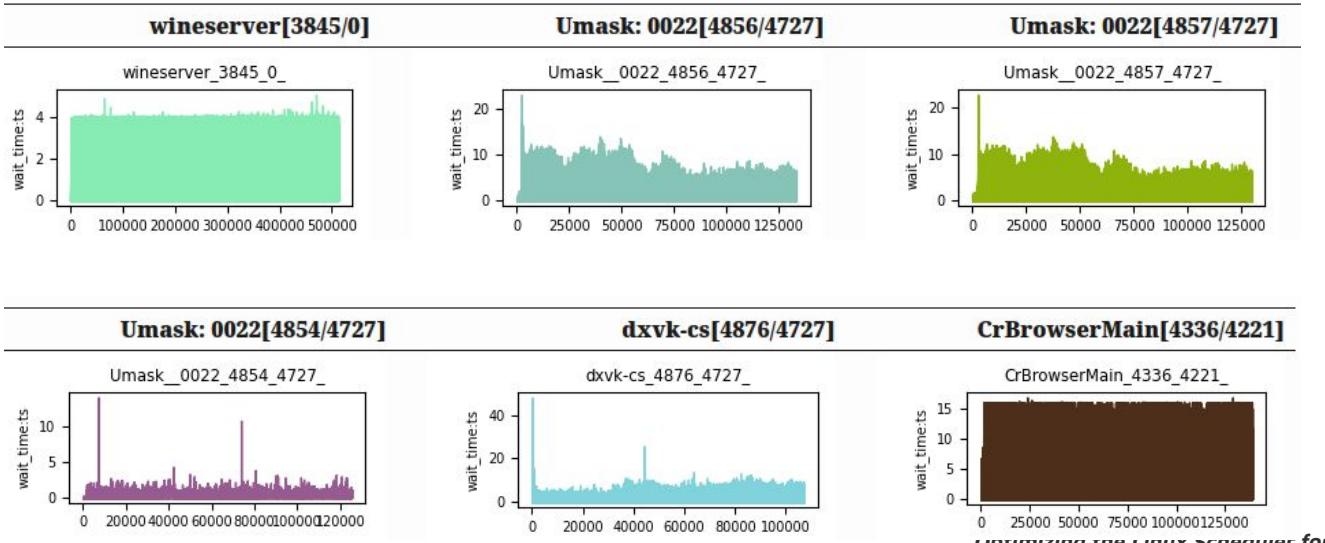


Call stacks which triggered scheduling in `a game`

Task waiting time

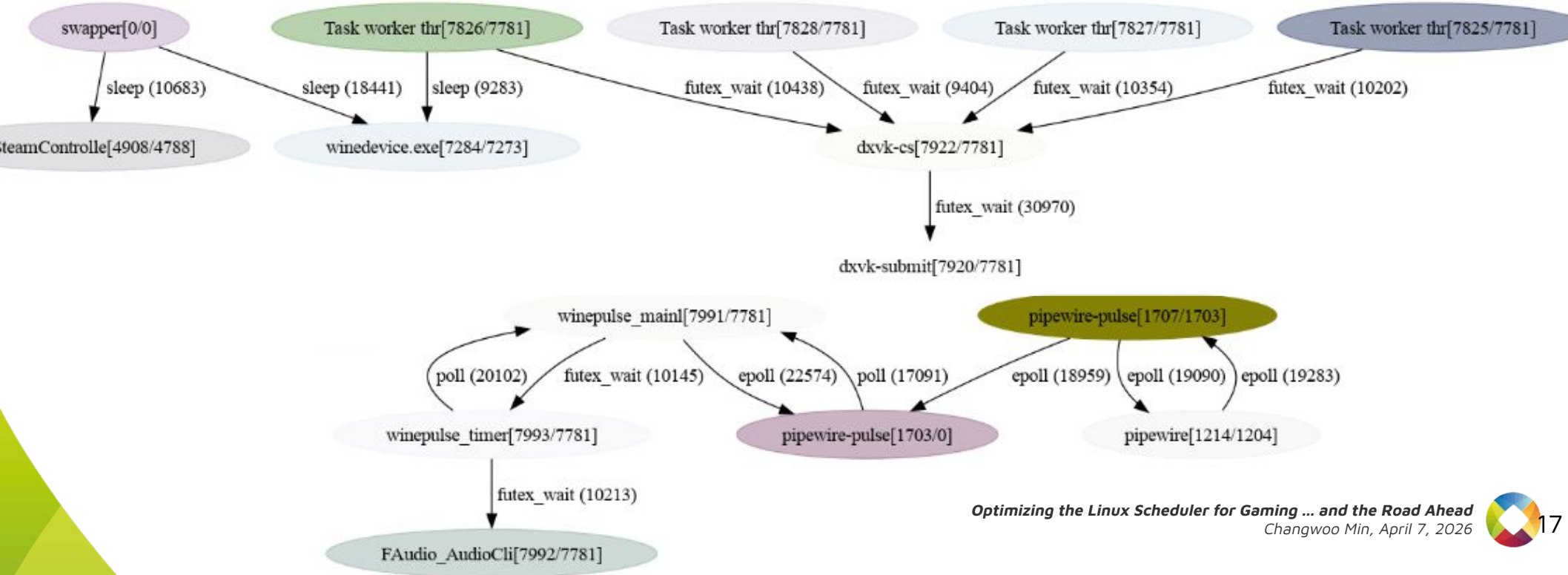
- Wait time from one schedule to another schedule is a task's behavioral property. **Each task's wait time is pretty constant.**

Wait times over time in a game



Tasks are tightly linked in a graph

- **Task graphs of top 50 percentile of waiter-wakers**



Ok, then does kernel allow a custom scheduler?

- **Yes, sched_ext is a BPF-based extensible scheduler framework.**
  - It is officially merged to the Linux mainline kernel at 6.12.
- **sched_ext enables scheduling policies to be implemented in a BPF program.**
- **The development procedure is somewhat similar to kernel module programming but BPF guarantees safety (e.g., no memory bugs);**

  (1) Write a scheduler policy in BPF.

  (2) Compile it.

  (3) Load it onto the system, letting BPF and sched_ext infrastructure do all of the heavy lifting to enable it.

sched_ext scheduler consists of three parts:

(1) The sched_ext core in kernel

- The kernel part is integrated with the scheduler core and provides hooks to redefine scheduling policy.
- E.g., `ops.enqueue()`, `ops.dispatch()`, `ops.select_cpu()`

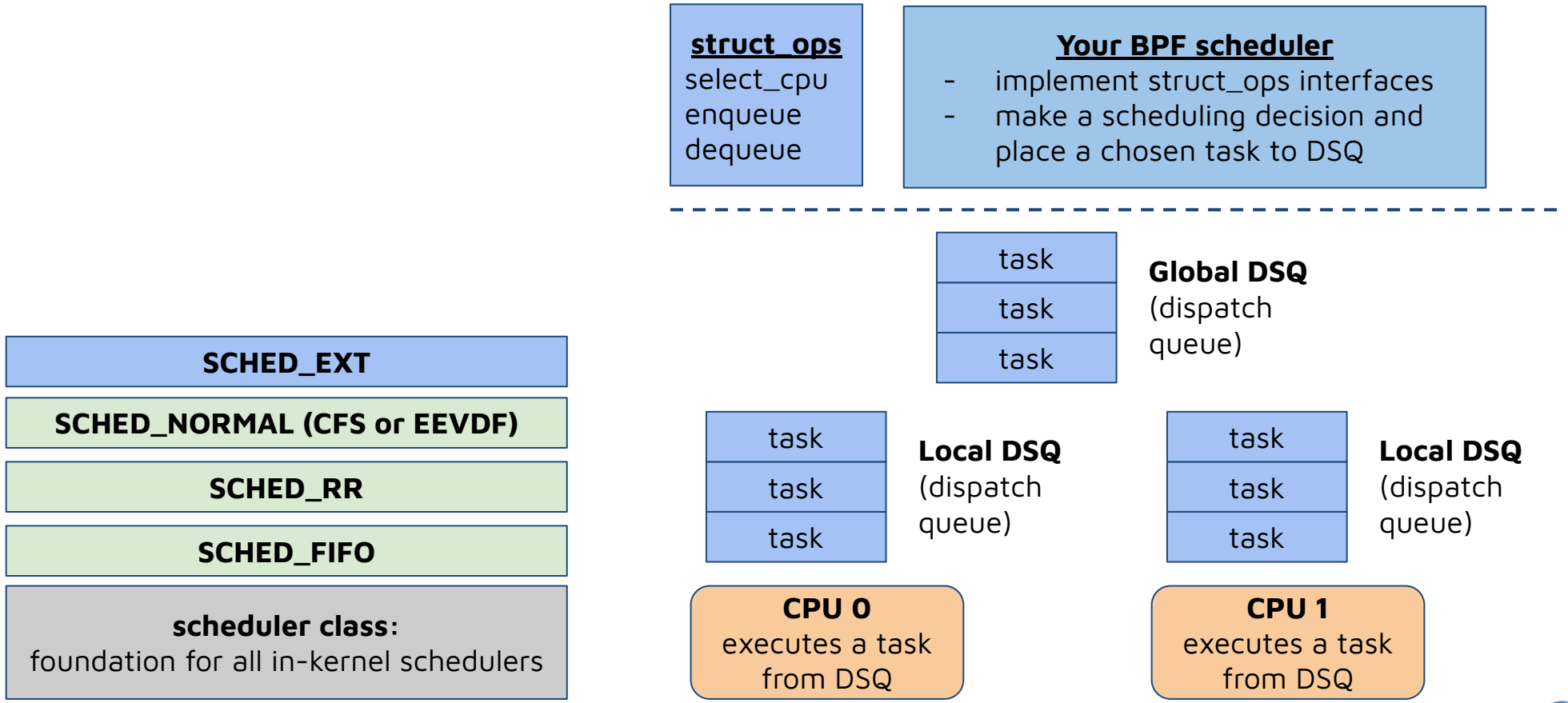
(2) The BPF part of a custom scheduler

- The BPF part defines the custom scheduling policy.
- The sched_ext core calls a callback written in BPF when custom scheduling policy is needed.
- For example,
  (a) What to do when a task becomes runnable so it needs to be enqueued?
  (b) What to do when a CPU needs to run a task among enqueued tasks?

(3) The user-space part of a custom scheduler

- Load and set up the BPF code.
- Usually thin and written in Rust.

Overall architecture of sched_ext



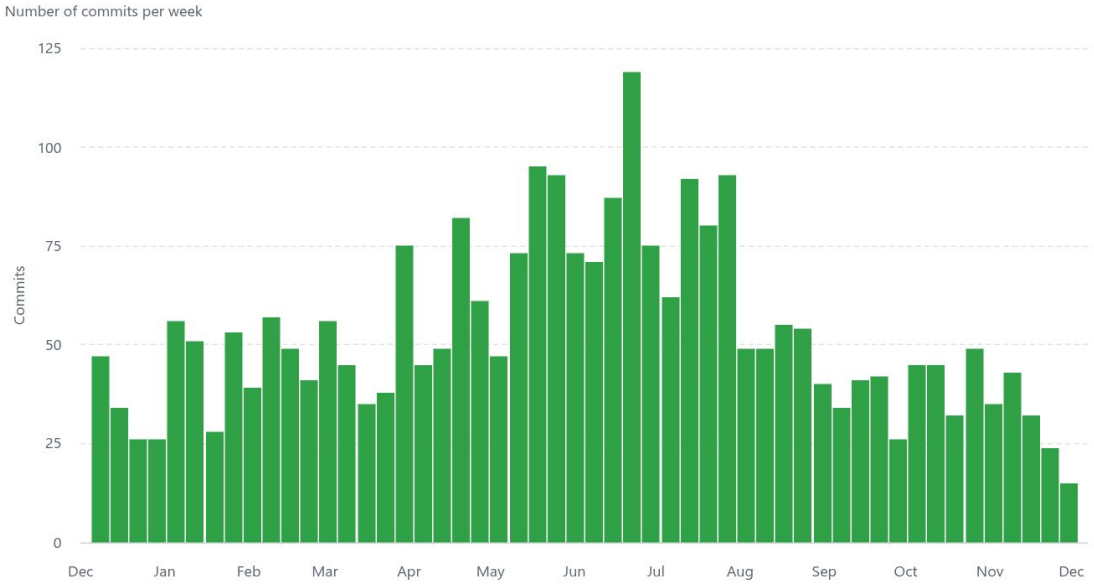
Implementing scheduling policies with sched_ext

- BPF program must implement a set of **callbacks**

```
struct sched_ext_ops {
    /** select_cpu - Pick the target CPU for a task which is being woken up */
    s32 (*select_cpu)(struct task_struct *p, s32 prev_cpu, u64 wake_flags);

    /** enqueue - Enqueue a task on the BPF scheduler */
    void (*enqueue)(struct task_struct *p, u64 enq_flags);

    /** dequeue - Remove a task from the BPF scheduler */
    void (*dequeue)(struct task_struct *p, u64 deq_flags);

    /** dispatch - Dispatch tasks from the BPF scheduler and/or consume DSQs */
    void (*dispatch)(s32 cpu, struct task_struct *prev);

    /** runnable - A task is becoming runnable on its associated CPU */
    void (*runnable)(struct task_struct *p, u64 enq_flags);

    /** running - A task is starting to run on its associated CPU */
    void (*running)(struct task_struct *p);

    /** stopping - A task is stopping execution */
    void (*stopping)(struct task_struct *p, bool runnable);

    /** quiescent - A task is becoming not runnable on its associated CPU */
    void (*quiescent)(struct task_struct *p, u64 deq_flags);

    /** yield - Yield CPU */
    bool (*yield)(struct task_struct *from, struct task_struct *to);
```

Custom scheduler development is very active

- **Sched_ext Schedulers and Tools**
  - https://github.com/sched-ext/scx/
- **Commits over the year**



So, what's your scheduler for gaming?

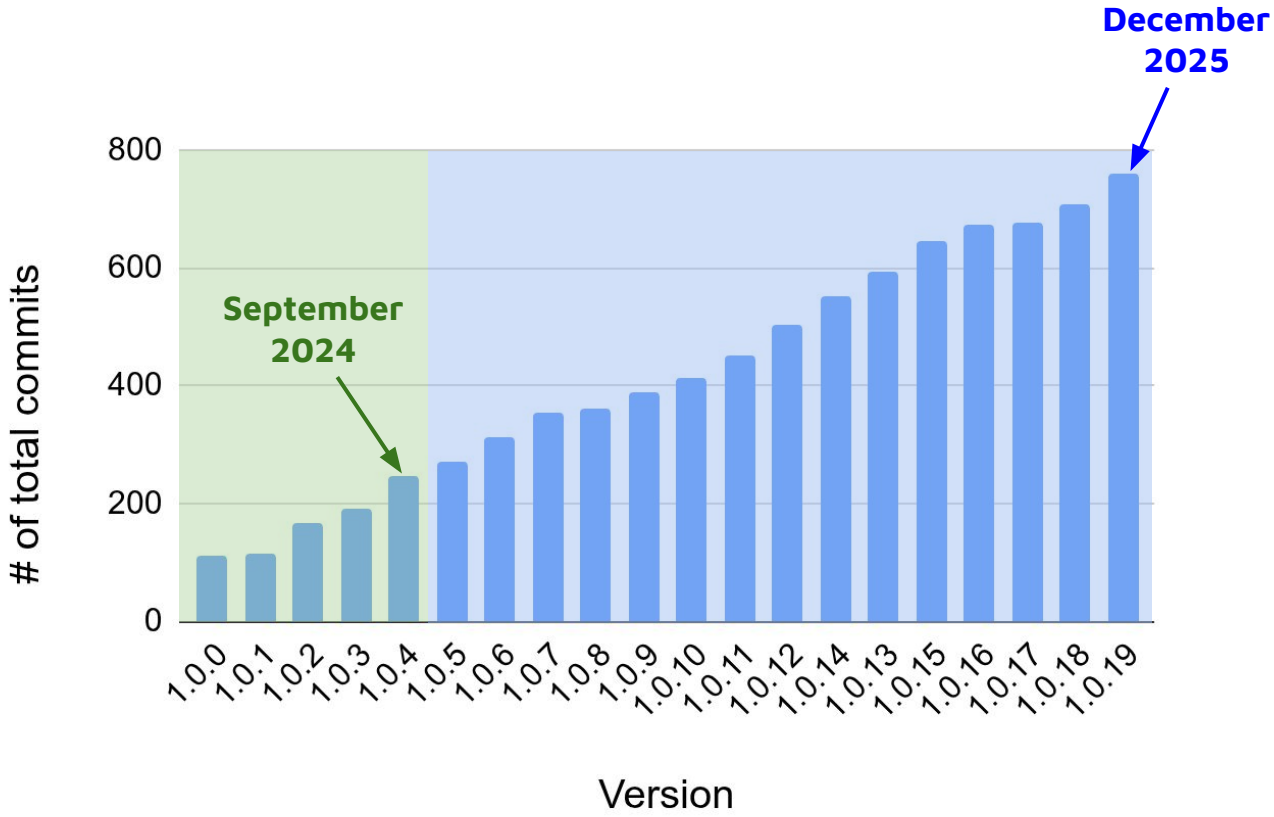
LAVD: Latency-criticality Aware Virtual Deadline scheduler

- **A new scheduling algorithm motivated by gaming workloads**
  - Implemented as a BPF scheduler on top of the sched_ext framework.
- **Virtual deadline-based, fair scheduling algorithm**
  - Task's scheduling urgency is represented as a task deadline.
    - A task with the earliest deadline will be chosen for execution.
    - Deadline is virtual, representing relative urgency of scheduling (not a wall clock time).
  - Respect nice value and pursue the fair use of CPU time.
- **Two main goals:**

(1) Prevent latency-spikes by urgently scheduling latency-critical tasks.

(2) Reduce energy consumption by wisely using hybrid CPUs (little/medium/big).

Development Status



LAVD should decide three things:

(1) Given a set of runnable tasks, which task should run first?

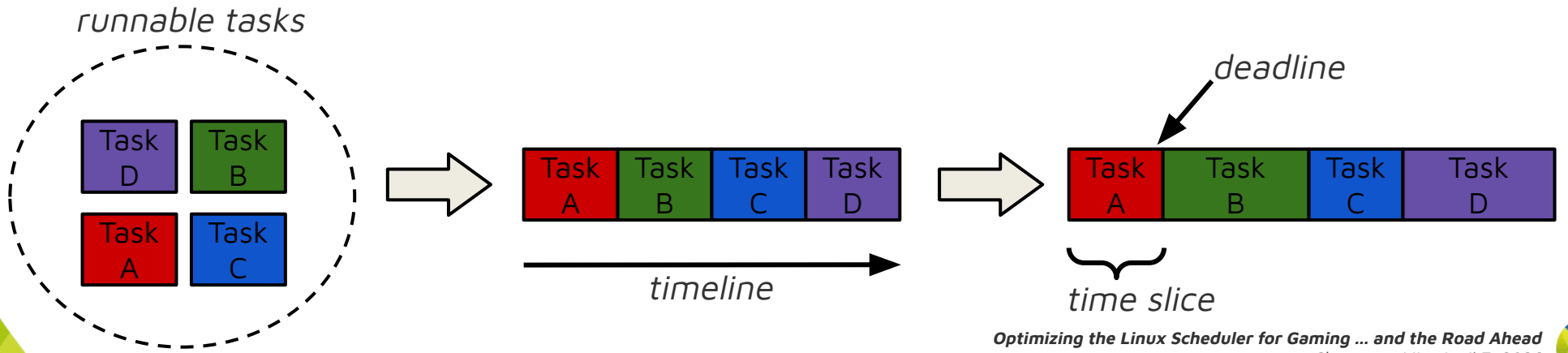
- How to decide task's virtual dealdine

(2) How long each task should run?

- How to determine task's time slice

(3) Where to run the task?

- How to choose a type of CPU to run (big/medium/little)



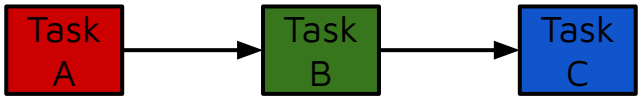
Overall procedure of LAVD

- LAVD is a deadline-based scheduling algorithm, so its overall procedure is similar to other deadline-based scheduling algorithms.
- Under LAVD, **a runnable task** has **its time slice and virtual deadline**, which are decided by the scheduler.
- The LAVD scheduler picks **a task with the earliest virtual deadline** and allows it to **execute for the given time slice**.

(1) Deciding task's virtual deadline

- **Schedule a latency-critical task first**
  - Assign a tighter deadline for a latency-critcial tasks.
  - Then, how to define a latency-criticality of a task?

Let's leverage the task graph!

  - 

  - **Wake up frequency**
    - Task A wakes up Task B, then Task B wakes up Task C.
  - **Wait frequency**
    - Task C waits for Task B, Task B waits for Task A.

(1) Deciding task's virtual deadline

- **Implications of wake-up/wait frequency**
  - High wake up frequency ⇒ important producer in a task graph
  - High wait frequency ⇒ important consumer in a task graph
  - Both are high ⇒ important task in the middle of a task chain
- **Virtual deadline of a task**
  - pipeline factor = f(wake freq, wait freq)
  - priority factor = f(nice priority)
  - fairness factor = f(vruntime)

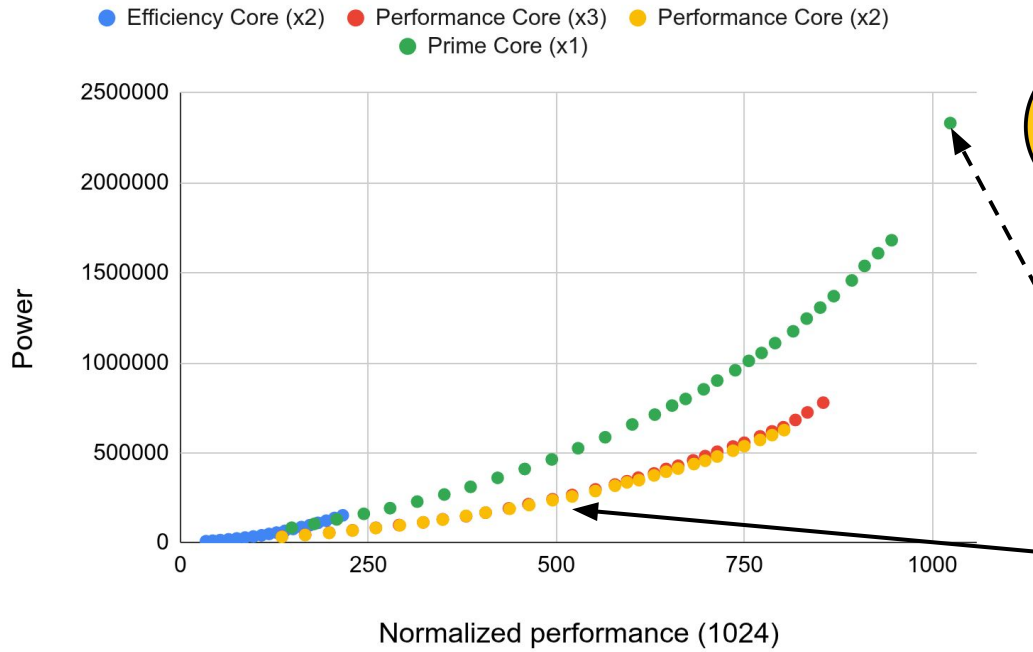
(2) Deciding task's time slice

- **We want to make each and every task run in a fixed time interval.**
  - A fixed time interval == targeted latency (e.g., 15 msec)
  - This helps ensuring the forward progress of all tasks without starvation.

- **Time slice = f(number of runnable tasks)**
  - More runnable tasks ⇒ shorter time slice for each task
  - Less runnable tasks ⇒ longer time slice for each task

(3) Deciding a type of CPU to run on

- **Different core types have different performance vs. energy consumption tradeoffs.**



Suppose we need the performance of 1024.
What is the best use of hybrid cores?

Answer 1: Let's use a single prime core!

Answer 2: Let's use two performance cores!

Energy Model Aware Scheduling (EMAS)

- **Key idea**
  - Given the required computing power, suggest a list of CPUs to meet the computing demand with minimum energy consumption.
- **For example,**
  - Q: I need computing power of 1024.
  - A: Use CPU 2 and 3 (performance core).
- **Key challenges**
  - How to get an energy model?
  - How to derive the recommended CPUs for a required computing demand?
  - How to assign a task to a proper core type?

Getting an energy model?

- **The kernel already provides the energy model:**

```
/sys/kernel/debug/energy_model
├── cpu0            # Efficiency core (x2)
│   ├── ps:1017600
│   ├── ...
│   └── ps:902400
├── cpu2            # Performance core (x3)
│   ├── ps:1075200
│   ├── ...
│   └── ps:960000
├── cpu5            # Performance core (x2)
│   ├── ps:1075200
│   ├── ...
│   └── ps:960000
└── cpu7            # Prime core (x1)
    ├── ps:1017600
    ├── ...
    └── ps:902400
```

- **In the future, the netlink interface will be used:**
  - PM: EM: Add netlink support for the energy model

Deriving suggested CPUs for a demand

- **This is a combinatorial optimization problem:**
  - For all possible sets of {CPU, utilization} pairs,
    - Computing power and energy consumption are determined.
  - For each computing power generated,
    - Choose the set of {CPU, utilization} pairs with minimum energy consumption.
- **Energy Model Optimizer (EMO) constructs this when loading LAVD.**
  - Heuristics based optimization for efficiency.

Example output of EMO

CPU utilization range	Capacity range	CPUs to be used
~ 5.3%	~ 300	0, 1
~ 20.2%	~ 1138	2, 3, 0, 1
~ 60.1%	~ 3386	2, 3, 4, 5, 6
~ 70.6%	~ 3977	2, 3, 4, 5, 6, 0
~ 80.1%	~ 4508	2, 3, 4, 5, 6, 0, 1
~ 100%	~ 5627	7, 2, 3, 4, 5, 6, 0, 1

Efficiency core (x2)
Performance core (x3)
Performance core (x2)
Prime core (x1)

Assigning a task to a proper core type

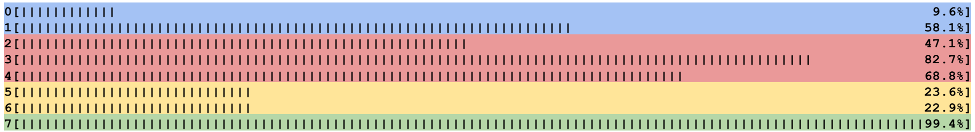
- **First, measure computing demand of a task**
  - At the high-level, same as PELT (Per-Entity Load Tracking) in kernel
  - Computing demand = *f(*task runtime, CPU capacity, CPU frequency*)*
- **Determine if a task is performance-critical or not**
  - Performance criticality = *f(*task's computing demand, wake-up/blocking frequency*)*
- **Top N% tasks in performance criticality are performance-critical tasks**
  - where N% is computing capacity that big cores provides.
- **When selecting a CPU of a task at ops.enqueue() and ops.select_cpu(),**
  - Try to match task's type (performance-critical or not) and CPU's type (big vs. little)

Experimental Results

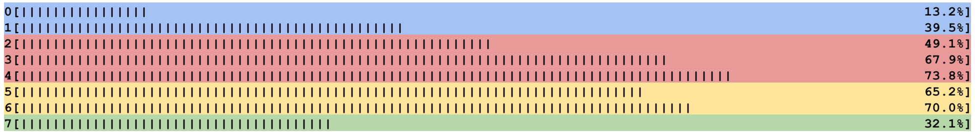
- **Demo on SteamDeck**

Experimental Results

- **On a Snapdragon development board.**
- **EEVDF**
  - FPS: around 50
  - CPU utilization



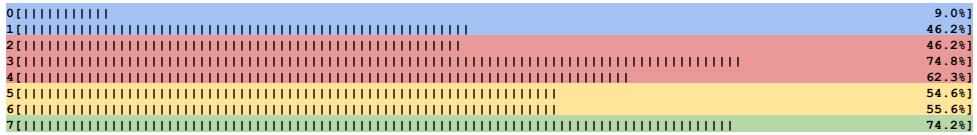
- **LAVD**
  - FPS: around 57, energy-rate: smaller than EEVDF
  - CPU utilization



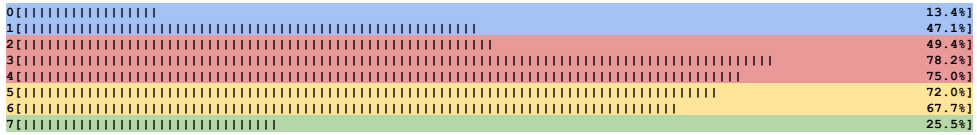
Efficiency core (x2)
Performance core (x3)
Performance core (x2)
Prime core (x1)

Experimental Results

- **On a Snapdragon development board.**
- **EEVDF**
  - FPS: around 50
  - CPU utilization



- **LAVD**
  - FPS: around 50, energy-rate: smaller than EEVDF
  - CPU utilization



Efficiency core (x2)
Performance core (x3)
Performance core (x2)
Prime core (x1)

… and Road Ahead

phoronix

ARTICLES & REVIEWS NEWS ARCHIVE FORUMS PREMIUM CONTACT CATEGORIES

Meta Is Using The Linux Scheduler Designed For Valve's Steam Deck On Its Servers

Written by Michael Larabel in Linux Kernel on 23 December 2025 at 06:10 AM EST. 29 Comments

An interesting anecdote from this month's Linux Plumbers Conference in Tokyo is that Meta (Facebook) is using the Linux scheduler originally designed for the needs of Valve's Steam Deck... On Meta Servers. Meta has found that the scheduler can actually adapt and work very well on the hyperscaler's large servers.

SCX-LAVD as the Latency-criticality Aware Virtual Deadline scheduler has worked out very well for the needs of Valve's Steam Deck with similar or better performance than EEVDF. SCX-LAVD has been worked on by Linux consulting firm Igalia under contract for Valve. SCX-LAVD has also seen varying use by the CachyOS Handheld Edition, Bazzite, and other Linux gaming software initiatives.

It turns out that besides working well on handhelds, SCX-LAVD can also end up working well on large servers too. The presentation at LPC 2025 by Meta engineers was in fact titled "*How do we make a Steam Deck scheduler work on large servers.*" At Meta they have explored SCX_LAVD as a "default" fleet scheduler for their servers that works for a range of hardware and use-cases for where they don't need any specialized scheduler.

At Meta they have explored SCX_LAVD as a "default" fleet scheduler for their servers that works for a range of hardware and use-cases for where they don't need any specialized scheduler.

Image credits

https://store.steampowered.com/steamdeck/
https://store.steampowered.com/sale/steammachine
https://store.steampowered.com/sale/steamframe
https://github.com/sched-ext
https://store.steampowered.com/app/2183900/Warhammer_40000_Space_Marine_2/
https://store.steampowered.com/app/253230/A_Hat_in_Time/
https://www.phoronix.com/news/Meta-SCX-LAVD-Steam-Deck-Server