

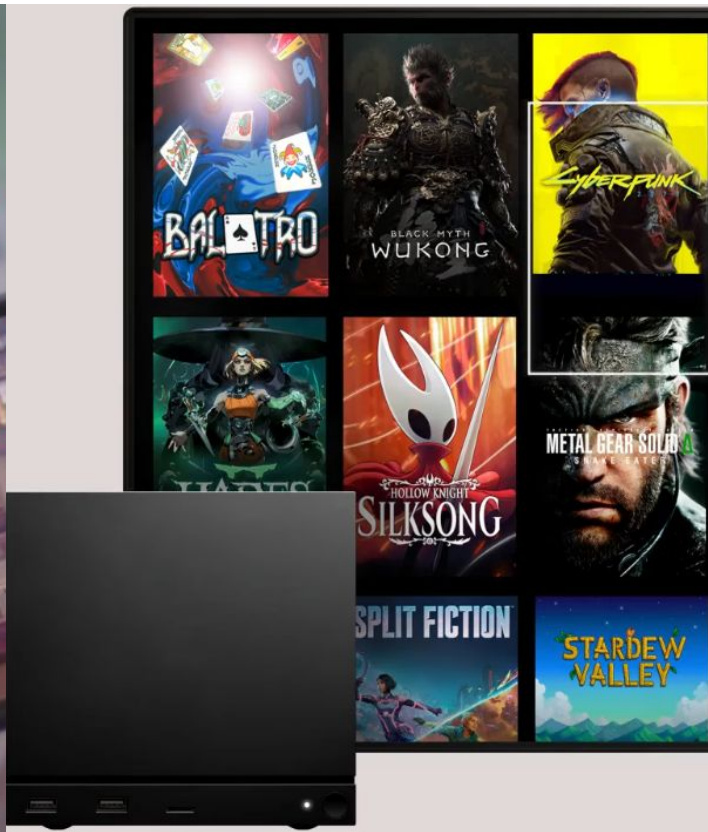
Optimizing the Linux Scheduler for Gaming and Large Servers

Changwoo Min
changwoo@igalia.com

April 27th, 2026



Gaming on Linux is here to stay



Gaming on Linux is here to stay

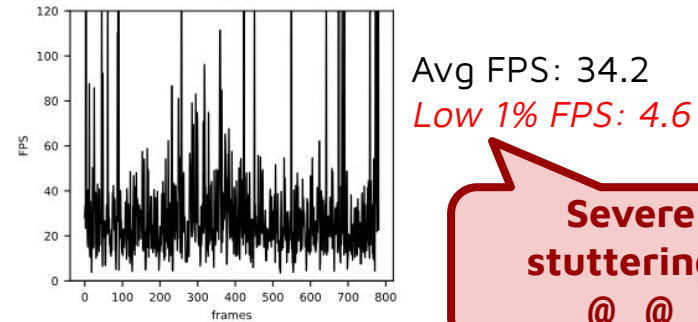
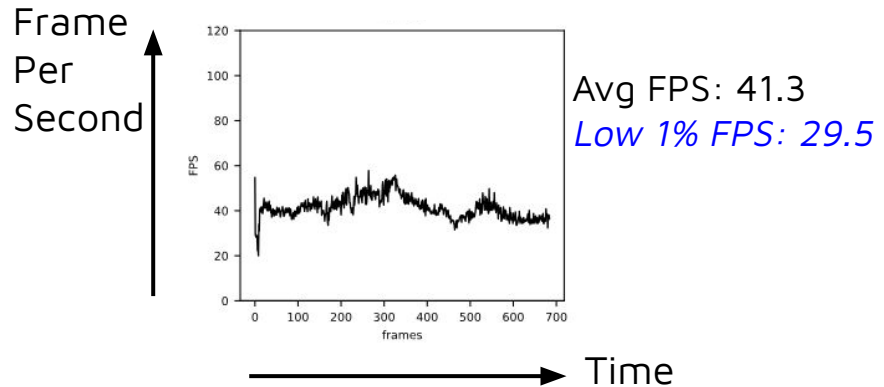
SteamOS runs x86 Windows games on Linux!

- **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



**Severe
stuttering!**
@_@

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



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

←  r/AMDHHelp • 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 games I played were Team Fortress 2 and Assassins creed Odyssey. I mentioned a few minutes the game starts stuttering and the fps get down which doesn't have with those games. My hardware consists of AMD 7, GeForce RTX 3090, mainboard Is B550Mx/e pro and I run the operating system windows 10. I also installed the driver for my GPU and it automatically updates drivers. I also did a BIOS update to fix this problem but nothing changed. Anyone an idea what problem should get fixed?

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

Optimizing the Linux Scheduler for Gaming and Large Servers
Changwoo Min, April 27, 2026



What is important to enhance gaming?

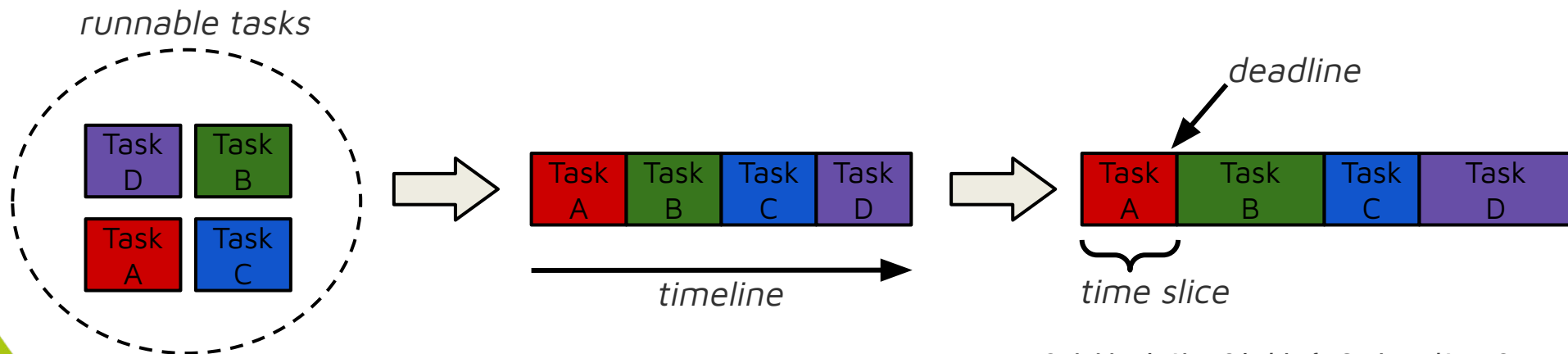
(2) Energy consumption

- 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

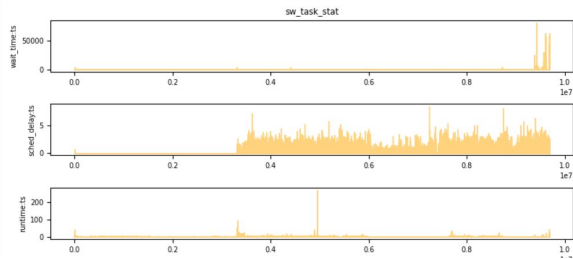
- 64-bit Architecture
- 1 Prime core, up to 3.4 GHz²
- 5 Performance cores, up to 3.2 GHz
- 2 Efficiency cores, up to 2.3 GHz



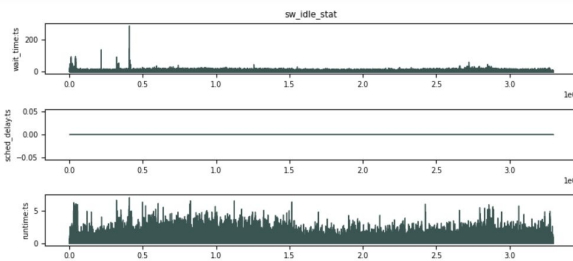
Analyzed a lot of scheduling traces

System-wide task and idle statistics

System-wide task stats data

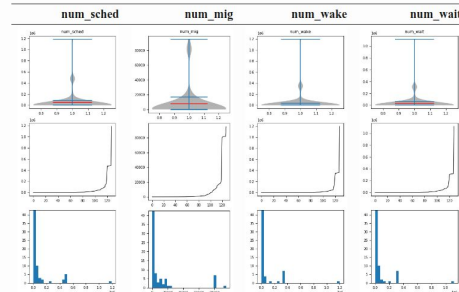


System-wide idle stats data



Task statistics using per-task average

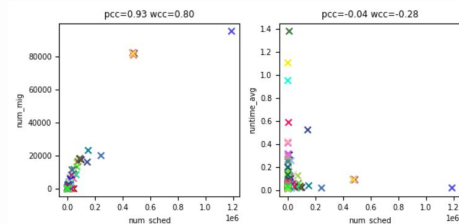
task_stats_data



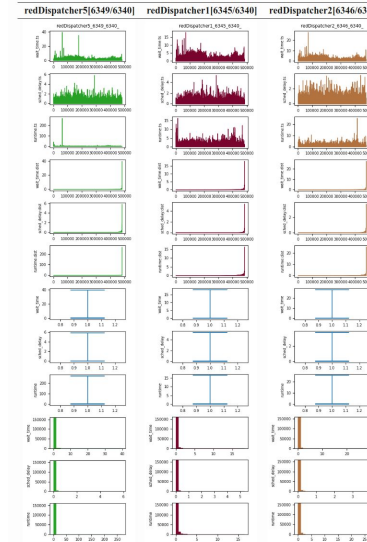
Correlation between task features

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

How useful is num_sched in predicting task's behavior?

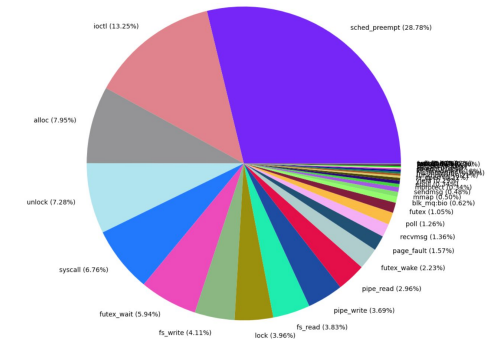


Task's wait_time, sched_delay, and runtimes

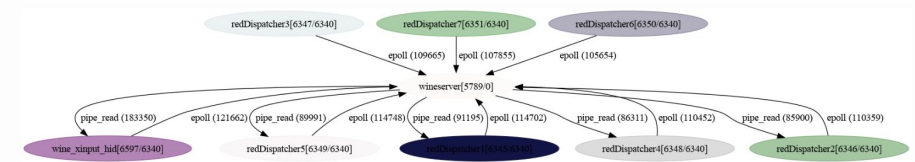


Callstacks triggered scheduling

- callsacks
- callsack_type



- 30.00 percentile of waker-waiter



Understanding gaming workloads

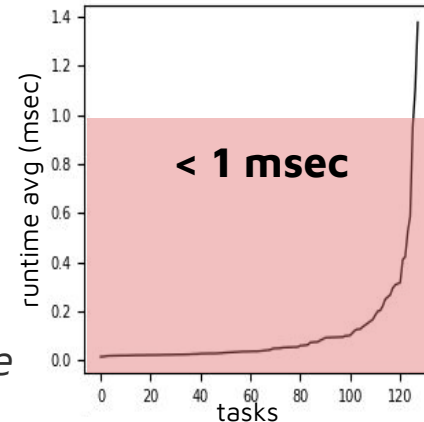
- **It is critical to understand the characteristics of gaming to design optimized scheduling policies for gaming.**
- **We collected and analyzed scheduling activities while playing games.**
 - VaporMark: <https://github.com/lgalia/vapormark>
- **In short, gaming workloads are:**
 - (1) [communication-intensive](#)
 - 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.



Task execution time per schedule

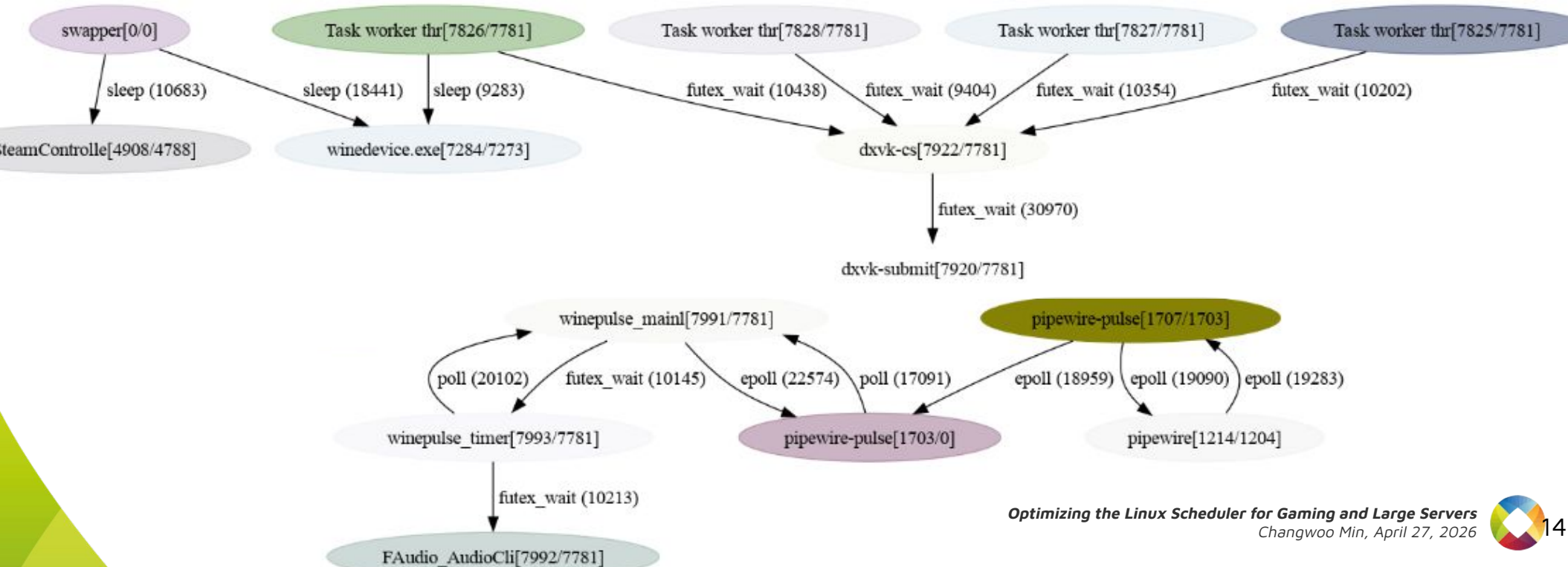
- In general, **tasks run for very short duration** – roughly **a few 100s usec** on average to **a few msec maximum**.
- **Task execution time** is very stable and is **predictable using its average**.
 - Some coordination tasks run very shortly (e.g., wineserver:260 usec) but some work tasks (e.g.,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

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



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.



So, what's your scheduler for gaming?

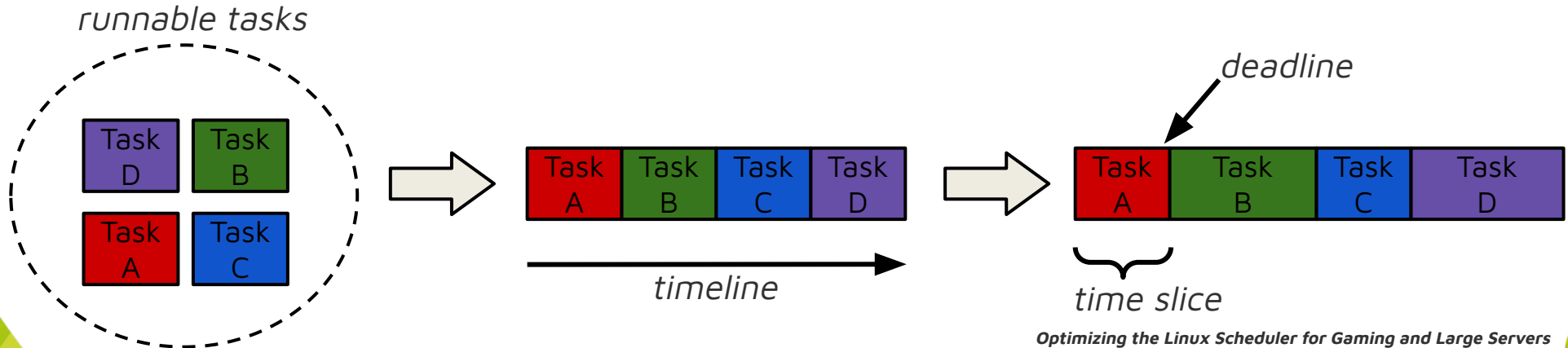
LAVD: Latency-criticality Aware Virtual Deadline scheduler

- **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).



LAVD should decide three things:

- (1) **Given a set of runnable tasks, which task should run first?**
 - How to decide task's virtual deadline
- (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

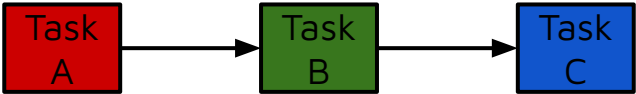
- LAVD is a deadline-based scheduling algorithm, so its overall procedure is similar to other deadline-based scheduling algorithms.
- Under LAVD, **a runnable task** has **its time slice and virtual deadline**, which are decided by the scheduler.
- The LAVD scheduler picks **a task with the earliest 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-critical tasks.
 - Then, how to define a latency-criticality of a task?



Let's leverage the task graph!

- 

```
graph LR; A[Task A] --> B[Task B]; B --> C[Task C];
```
- **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 \Rightarrow important producer in a task graph
 - High wait frequency \Rightarrow important consumer in a task graph
 - Both are high \Rightarrow important task in the middle of a task chain
- **Virtual deadline of a task**
 - pipeline factor = $f(\text{wake freq, wait freq})$
 - priority factor = $f(\text{nice priority})$
 - fairness factor = $f(\text{vruntime})$

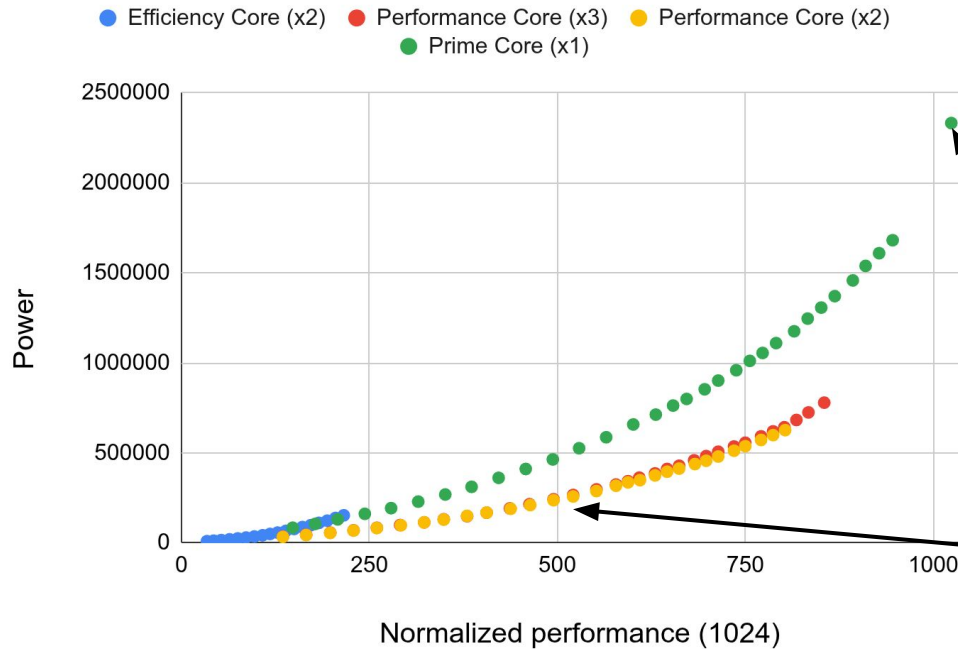


(2) Deciding task's time slice

- **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 \Rightarrow shorter time slice for each task
 - Less runnable tasks \Rightarrow 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?

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	CPU IDs	Core Types
~ 5.3%	~ 300	0, 1	Efficiency core (x2)
~ 20.2%	~ 1138	2, 3, 0, 1	Performance core (x3) Performance core (x2)
~ 60.1%	~ 3386	2, 3, 4, 5, 6	Prime core (x1)
~ 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	

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 = $f(\text{task runtime, CPU capacity, CPU frequency})$
- **Determine if a task is performance-critical or not**
 - Performance criticality = $f(\text{task's computing demand, wake-up/blocking frequency})$
- **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)

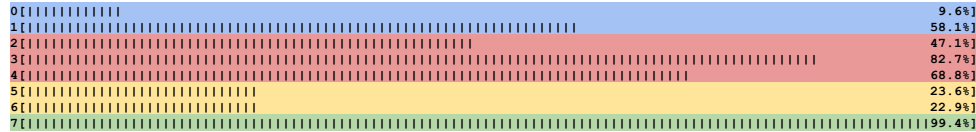
Experimental Results

- Demo on SteamDeck

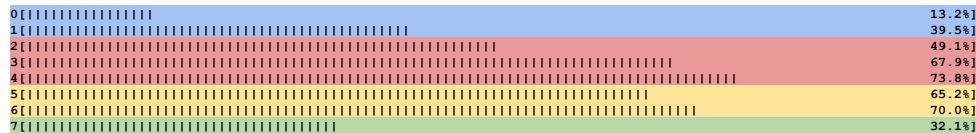


Experimental Results

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



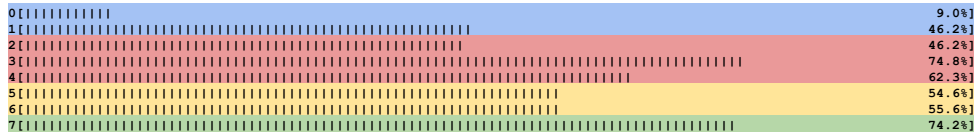
- LAVD
 - FPS: around 57, energy-rate: smaller than EEVDF
 - CPU utilization



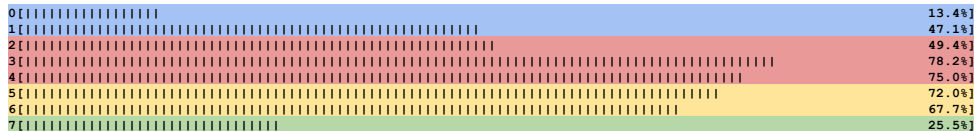
Efficiency core (x2)
Performance core (x3)
Performance core (x2)
Prime core (x1)

Experimental Results

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

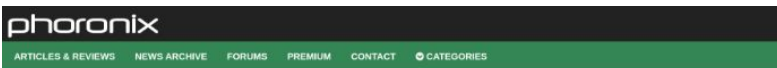


- **LAVD**
 - FPS: around 50, energy-rate: smaller than EEVDF
 - CPU utilization



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

... and Large Servers



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.



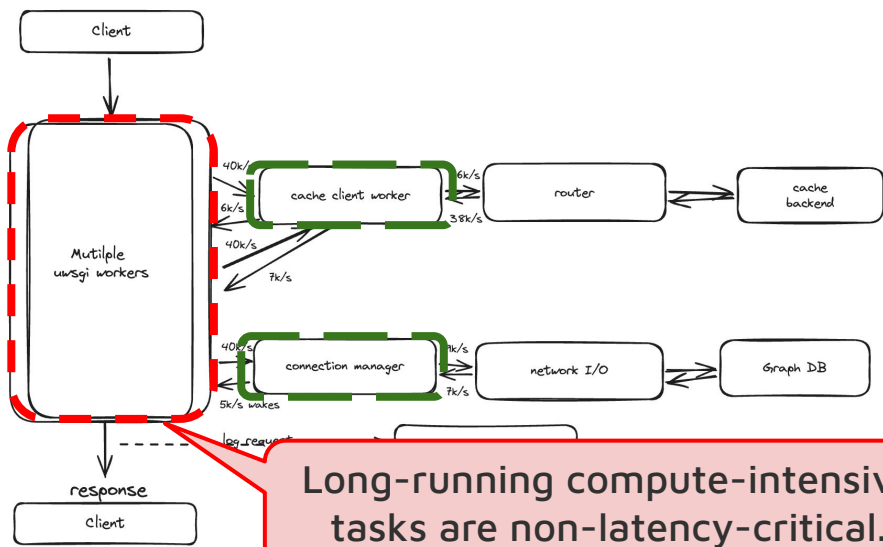
What are different in data center?

- **Hardware are different.**
 - A typical server machine == a large multi-LLC machine
 - 1 or 2 socket machine
 - 1 socket = 12 LLC domains (e.g., AMD CCX)
 - 1 LLC domain = 8 cores
- **Workloads are different too.**
 - Not all tasks are short-running.
 - L3 cache locality is critical in performance.
 - Some workloads are very IRQ-heavy.
 - Still, latency, especially tail latency, is the key performance metric.

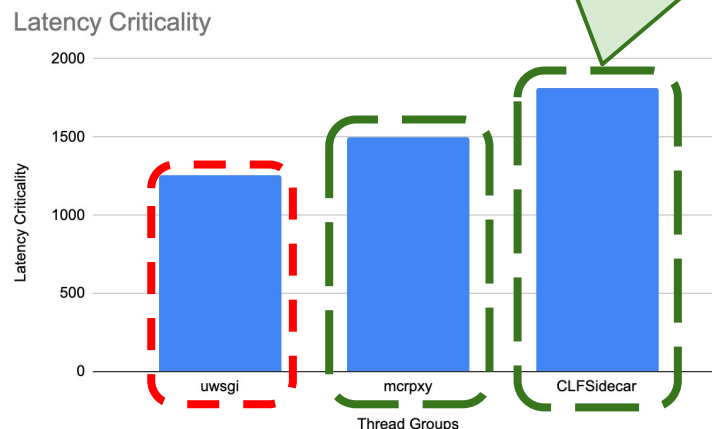


LAVD detects latency-critical tasks

- **Waker/wakee frequency detects latency-critical tasks pretty well.**
 - Urgently schedule latency-critical tasks and reduce tail latency.
- **Case study: a user-facing web service**



Short-running, wake-up heavy tasks are latency-critical.



Not all tasks are short-running

- **Some long-running tasks are in the middle of a task chain.**
 - Long-running tasks can be preempted out when its time slice is exhausted.
 - This delays the completion of long-running tasks' job.
 - And, the dependent wakee tasks will be also delayed.
 - This causes cascading delay and high tail latency.
- **Our solution: time slice boost**
 - If a task is long-running and latency-critical,
 - assign a longer time slice than the default slice proportionally to its latency criticality.



L3 cache locality is critical

- **Multiple LLC domains + large cache footprint workloads**
 - Tasks' cache footprint is large, so task migration across LLC domain causing L3 miss is expensive.
⇒ A task should stick to the LLC domain.
 - A machine has multiple LLC domains, so ideally all LLC domains should be equally used for work conservation.
⇒ A task should be moved quickly to a lower utilized LLC domain.
- **Our solution: budget-controlled cross-domain load balancer**
 - Cross-domain load-balancer
 - Task migration within a domain is almost free (large cache footprint ⇒ not worth to pursue L1/L2 cache locality).
 - Use per-domain shared runqueue for work conservation.
 - Budget-controlled load-balancer
 - To avoid the thundering herd problem, should not aim to completely equalize load across domains.
 - Instead, migrate tasks to reduce the half of imbalance in one interval (100 msec).



Some workloads are IRQ-heavy

- **Every homogeneous CPUs are not equal in capacity.**
 - There are IRQ-heavy workloads (e.g., network-intensive KV cache).
 - For performance, IRQ are pinned to a subset of CPUs.
 - For those CPUs, IRQ takes often more than 30% of CPU time.
 - Hence, less CPU time left to run fair tasks.
- **Our solution: latency criticality-aware task placement**
 - If a latency-critical task is interrupted, the tail latency surges.
 - Let's place latency-critical tasks to less IRQ-heavy CPUs.



Some workloads are IRQ-heavy

- **Every homogeneous CPUs are not equal in capacity.**
 - There are IRQ-heavy workloads (e.g., network-intensive KV cache).
 - For performance, IRQ are pinned to a subset of CPUs.
 - For those CPUs, IRQ takes often more than 30% of CPU time.
 - Hence, less CPU time left to run fair tasks.
- **Our solution: latency criticality-aware task placement**
 - If a latency-critical task is interrupted, the tail latency surges.
 - Let's place latency-critical tasks to less IRQ-heavy CPUs.



Experimental Results

- Caching service: tail latency of GET and DEL operations

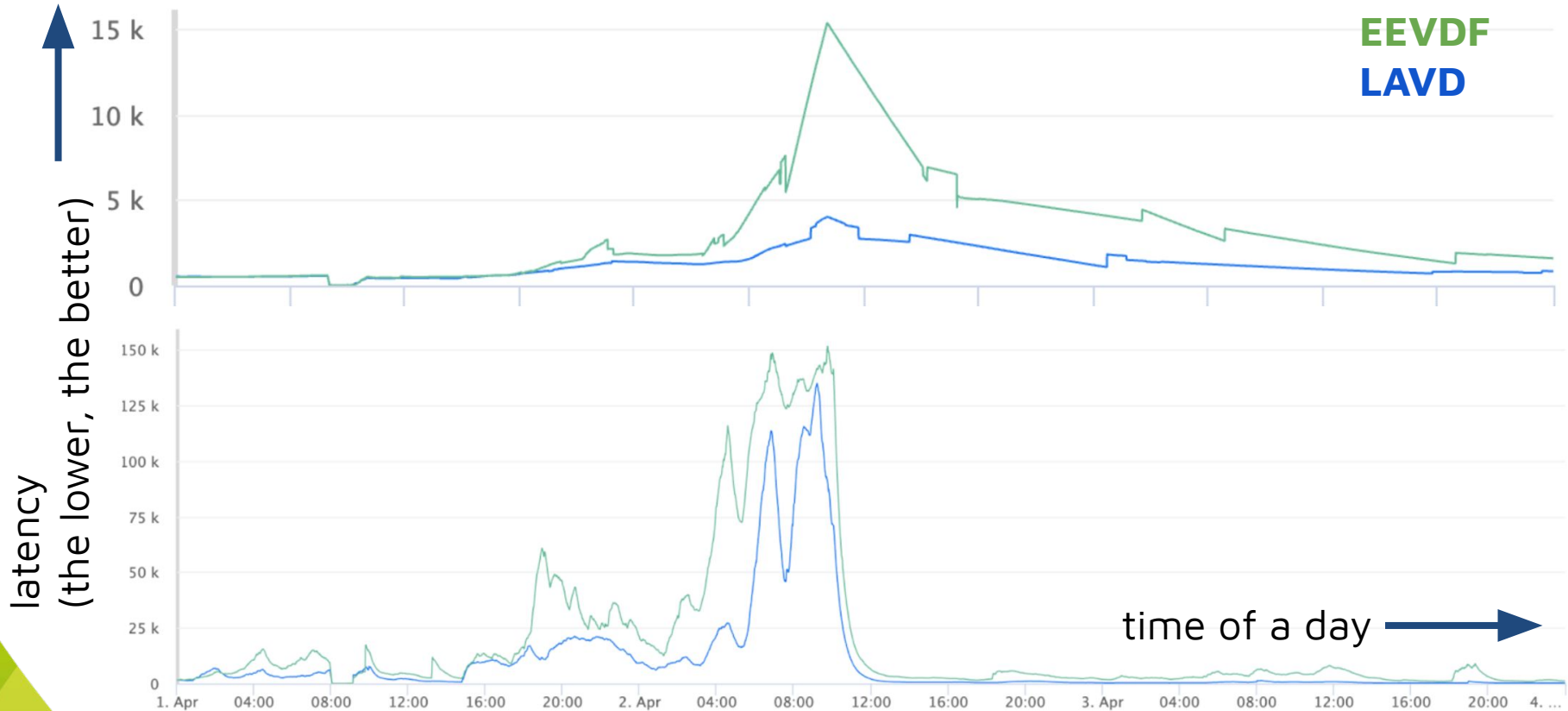




Image credits

<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>