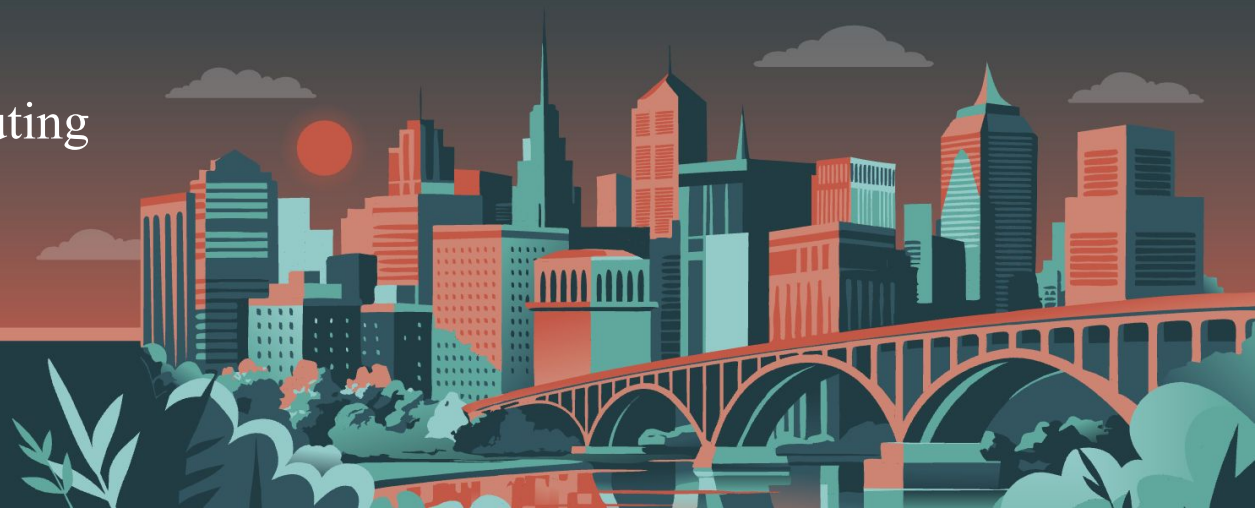
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

The Hidden Cost of Sleep: How Scheduler Wakeup Latency Impacts High Throughput AI Inference

Shubhang Kaushik Prasanna Kumar

Software Engineer @ Ampere Computing

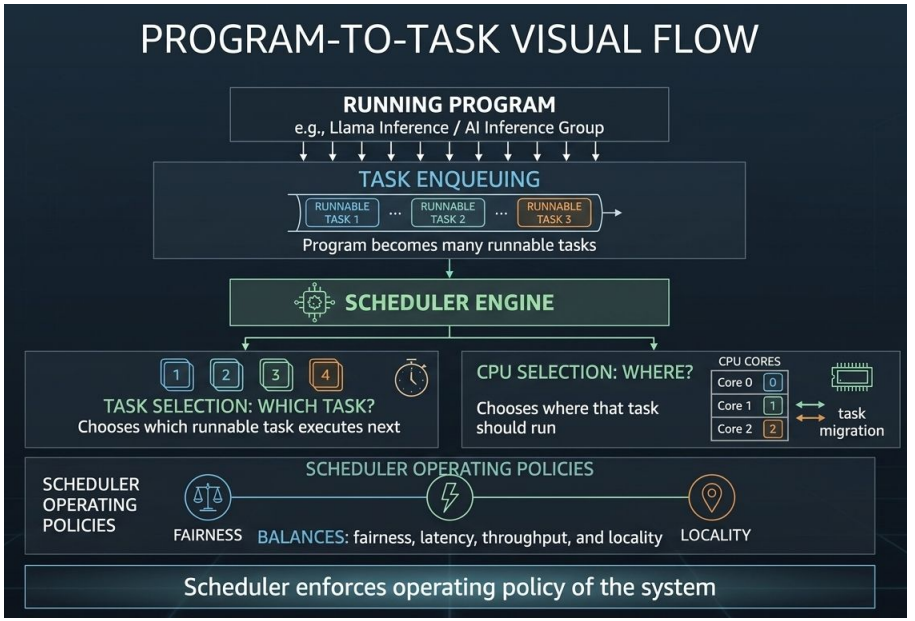


Agenda

- Introduction to Linux schedulers and the main scheduler classes
- Why hackbench, schbench, and cyclictest matter for scheduler optimization
- How AI inference differs from classical workloads
- CFS vs. EEVDF and the problem of sleep credit
- Scheduler operations and the wakeup path
- Stutter, avg_idle accounting, the fix, and workload impact

What Linux Scheduler Does

- Chooses which runnable task executes next
- Chooses where that task should run
- Balances fairness, latency, throughput, and locality
- Preempts tasks when a more eligible task appears
- Coordinates load balancing across cores and domains



Linux Scheduler Classes & where they apply

- **Stop Class (stop_sched_class)**: Highest priority for critical kernel movements, including migration threads and CPU hotplugging.
- **Deadline Class (dl_sched_class)**: Hard real-time SCHED_DEADLINE tasks with guaranteed CPU bandwidth and strict timing.
- **Real-Time Class (rt_sched_class)**: Fixed-priority SCHED_FIFO and SCHED_RR tasks for latency-sensitive apps like audio or drivers.
- **Fair Class (fair_sched_class)**: General workloads (currently managed by EEVDF, formerly CFS) prioritized by Eligibility and Virtual Deadlines rather than just vruntime.
- **Idle Class (idle_sched_class)**: Lowest priority; runs the swapper task (PID 0) only when the CPU is empty to save power.

Hackbench: Wakeups, IPC, and Scheduler Scalability

- Many communicating tasks repeatedly wake each other
- Stresses task creation, wakeup, enqueue, and load distribution
- Useful for spotting scheduler overhead at higher core counts
- A good proxy for bulk task churn and wake up heavy behavior

Run hackbench with 20 groups, 1000 loops per task, using processes

> hackbench -g 20 -l 1000 -p

Schbench: Tail Latency Under Lock Handoff Pressure

- Models wakeup behavior around mutex contention
- Targets lock handoff latency and scheduler-induced queueing
- Very sensitive to tail behavior and outliers
- Useful when averages look fine but users still see spikes

Run schbench with 16 message-passing worker threads

> schbench -m 16 -t 2 -c 1000

-m: number of message-passing groups (concurrency)

-t: number of worker threads per group

-c: total number of messageops to perform

Cyclictest: Measuring Scheduling Jitter

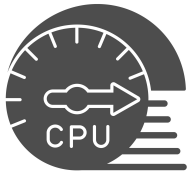
- Measures timer wakeup latency and worst-case delay
- A standard tool for studying deterministic behavior
- Highlights jitter introduced by interrupts, preemption, and scheduling delay
- Useful baseline for ‘how noisy is the system’

Run cyclictest on all 80 cores with high priority

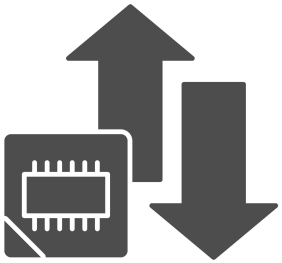
> sudo cyclictest -a -t -p 80 -n -l 100000

-a: use all CPUs , -t: create one thread per CPU , -p 80: run with a priority of 80, # -n: use clock_nanosleep for high precision

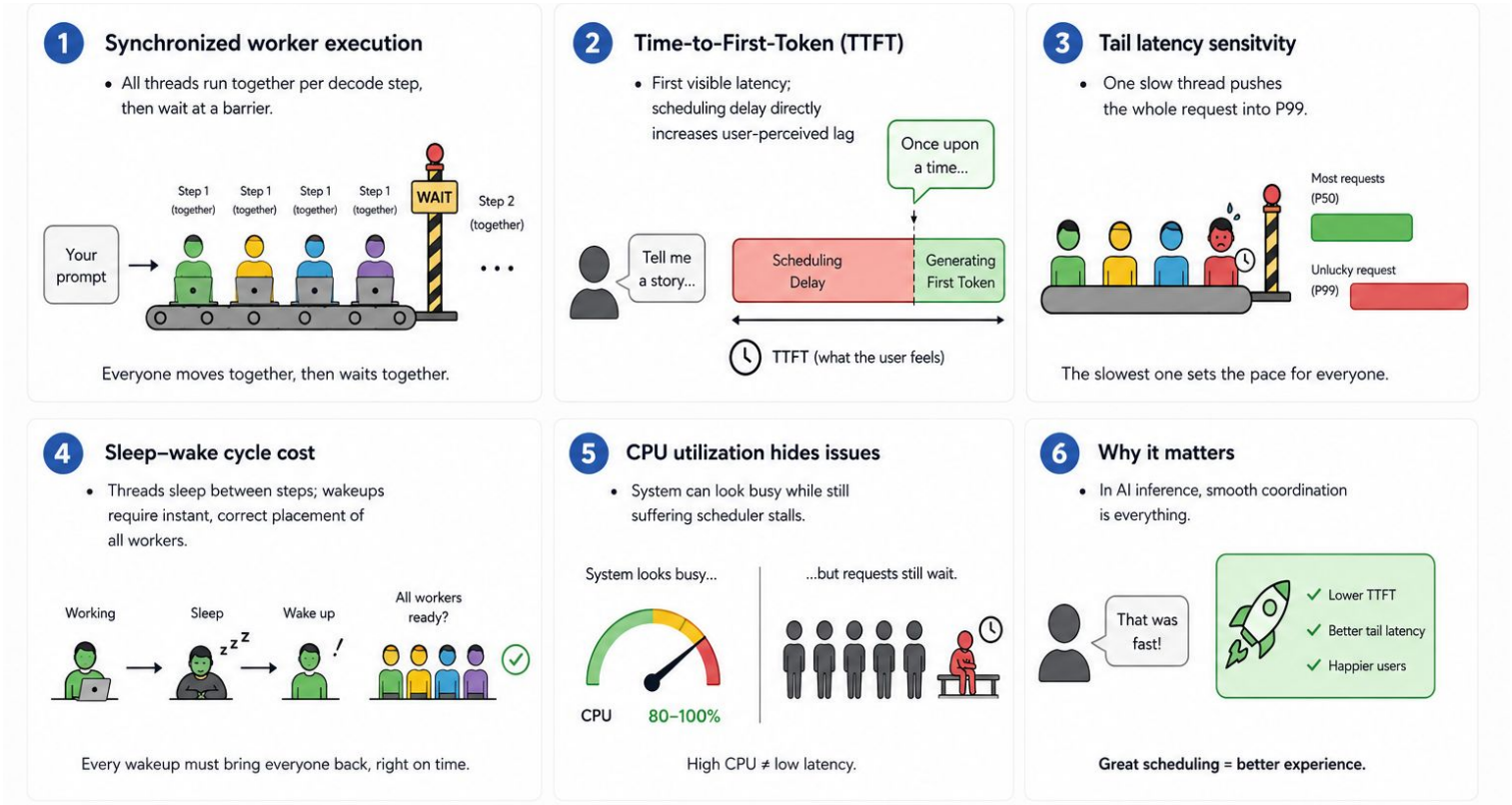
What These Benchmarks Tell Us Together



- Hackbench: wakeup throughput and scalability
- Schbench: lock handoff and tail latency
- Cyclictest: jitter and worst-case delay
- Together they predict where synchronized AI workloads will hurt

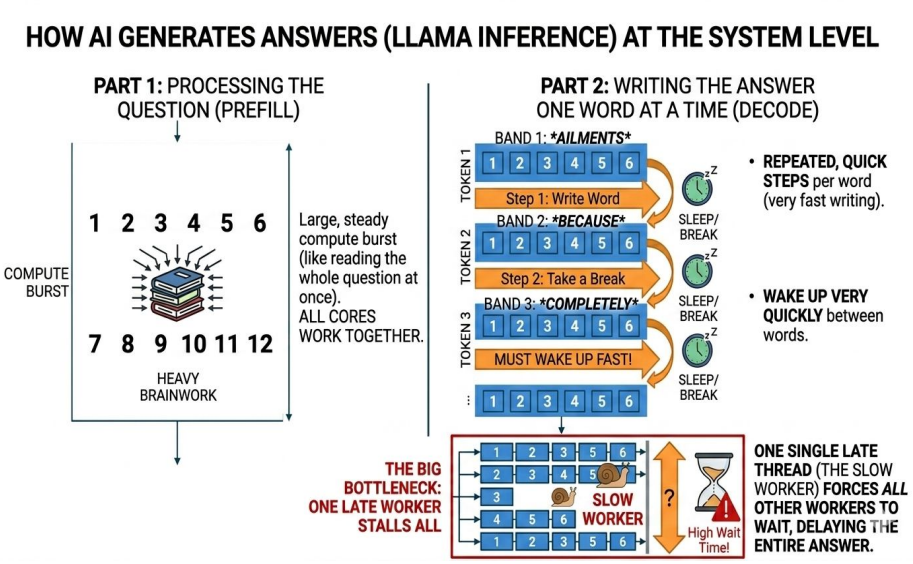


Why AI Inference Is Different



How Llama Inference Behaves at the System Level

- **Wake → compute → wait :** Repeats for every token generation step
- **Prefill**: large, more regular parallel compute burst
- **Decode**: repeated short synchronized steps per token
- Workers sleep between rounds and must wake quickly
- A single late thread expands the barrier wait for all



What AI Inference Is Looking For From the Scheduler

- Fast wakeup and fast enqueue of runnable workers
- Correct CPU placement with low migration cost
- Warm-cache reentry and NUMA-aware locality
- Low jitter and low interference from background work
- Tight alignment of worker completion times

The goal is not only fairness, it is fast wakeup, correct placement, low noise, and tight worker alignment.

CFS vs. EEVDF: The Policy Shift

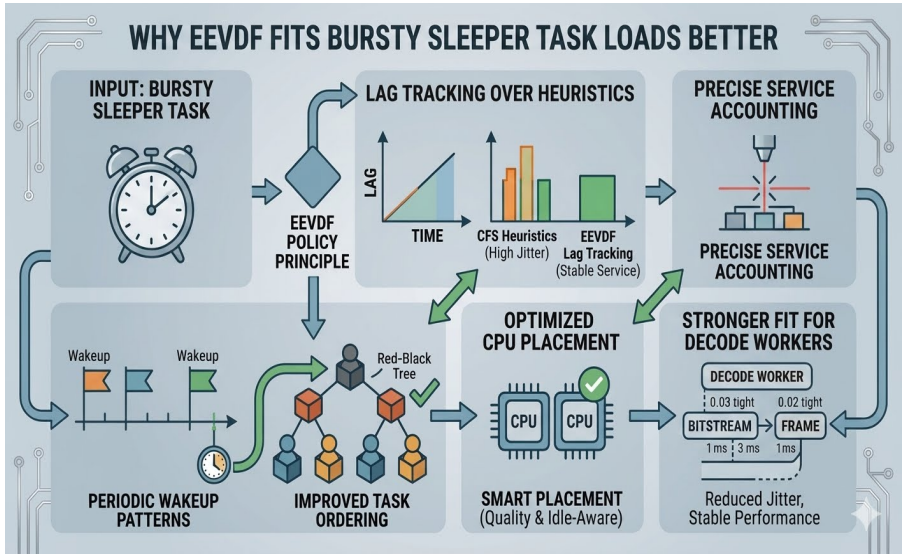
CFS - Completely Fair Scheduler
EEVDF - Earliest Eligible Virtual Deadline First

	CFS	EEVDF
Scheduling Basis	Virtual runtime (vruntime)	Eligibility + virtual deadline + lag
Decision Method	Smallest vruntime runs next	Earliest eligible virtual deadline runs
Sleep/Wake Behavior place_entity()	Heuristic credit on wake-up (nice value)	Tracks exact owed time using lag
Fairness Model	Equal CPU over time window	Explicitly tracks owed CPU time
Key Weakness	Can mis-handle bursty sleep/wake workloads	Handles sleep/wake patterns more precisely
Best Fit	General-purpose workloads	Latency-sensitive workloads (e.g., AI inference)

Why EEVDF Fits Bursty Sleepers Better

EEVDF improves policy treatment of sleepers, but placement quality and idle accounting still matter.

- Tracks lag instead of relying only on reentry heuristics
- Better reflects service owed to sleeping tasks
- Improves ordering for periodic wakeup-heavy patterns
- A stronger fit for decode-style worker behavior



SPECjbb® 2015 & The EEVDF Transition Analysis

What is **SPECjbb® 2015**?

The Industry Standard: Evaluates server-side Java performance by simulating a massive retail infrastructure (POS, online orders, and data mining).

Two Critical Metrics:

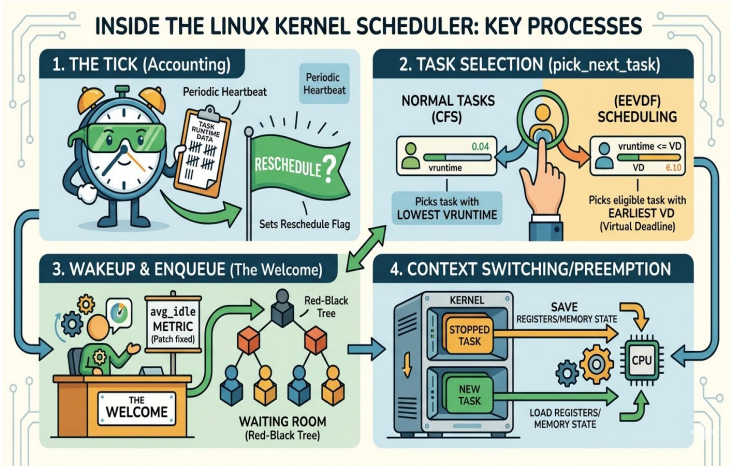
- **max-jOPS**: Absolute peak throughput (how much work can the system do?).
- **critical-jOPS**: Throughput maintained under strict latency SLAs (how fast can it respond under load?.

The shift from CFS to EEVDF (Linux 6.6+) changes how Java's bursty, lock-heavy threads are handled:

- **The Eligibility Bottleneck**: EEVDF's "eligibility" math can micro-stall bursty Java threads (GC/JIT) if they have negative lag, collapsing critical-jOPS metrics.
- **Virtual Deadline Jitter**: Stale avg_idle signals lead to poor placement of threads with tight virtual deadlines, causing immediate SLA failures. (estimated)
- **Lock-Holder Preemption**: EEVDF's aggressive preemption can deschedule a thread while it holds a mutex, triggering cascading stalls across the JVM.
- **Tuning Shift**: Traditional CFS fairness knobs are replaced by EEVDF base_slice logic, requiring a complete re-tuning of Java thread fragmentation.

Linux Scheduler Operations

- **The Tick**: A periodic kernel "heartbeat" that updates the current task runtime & sets a "reschedule" flag if the task has used its fair share or a more important task is waiting.
- **Task Selection**: The pick_next_task() function selects the next process based on priority classes; for normal tasks (CFS) - it picks the one with the lowest virtual runtime (vruntime) ; (EEVDF) selects the eligible task (vruntime <= VD) with the earliest virtual deadline.
- **Wakeup & Enqueue**: When a task resumes, the scheduler selects a target CPU (using the avg_idle metric) & places the task into the "waiting room" (Red-Black Tree).
- **Context Switching**: The physical transition where the kernel saves the registers/memory state of the old task & loads those of the new one to begin execution.



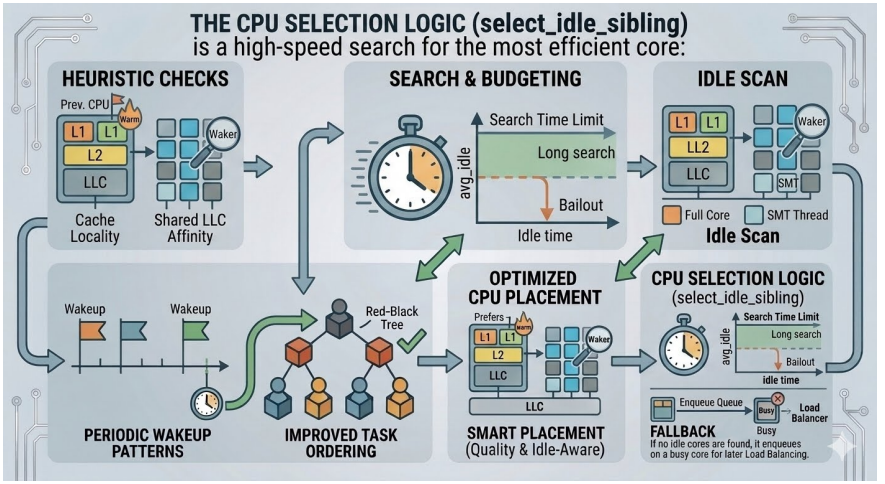
Scheduler Wake Up Path

The scheduler wake up path is the "re-entry" protocol for a thread transitioning from a sleeping state back to a ready-to-run state.

- **The Decision (select_task_rq)**: The kernel selects the optimal CPU core for the task, aiming for an idle one to maintain balance. This is where my patch uses avg_idle to prevent dumping tasks onto "fake idle" cores.
- **The Activation (ttwu_do_activate)**: The task is officially added to the target core's runqueue. It syncs the CPU clock, enqueues the task in the Red-Black Tree, and calculates its fair starting position (vruntime).
- **The Preemption Check (check_preempt_curr):** The scheduler determines if the waking task is important enough to immediately replace the currently running one. If so, a context switch is triggered; otherwise, the task waits its turn.

CPU Selection Logic - select_idle_sibling

- **Cache Locality**: Prefers the Last CPU used to leverage "warm" L1/L2 data.
- **Wake-Affinity**: Favors the Waker's CPU to keep related tasks within the same Last Level Cache (LLC).
- **Idle Scan**: Searches the LLC domain for an Idle Core or an idle SMT thread (hyperthread).
- **Search Budgeting**: Uses avg_idle to limit search time; if the system is busy, it bails out early to save overhead.
- **Fallback**: If no idle cores are found, it enqueues on a busy core for later Load Balancing.



Scheduler Wakeup Latency

Scheduler Wakeup Latency is the delay between a task being signaled to run and its actual execution on a CPU.

- **Selection Latency**: Time spent searching for an optimal core; stale data causes "search exhaustion."
- **Queuing Latency:** Time spent waiting in the Runqueue due to bad placement on a busy core.
- **The Straggler Effect**: A single delayed thread stalls the entire 80-core team at the next synchronization barrier.
- **Performance Impact**: Increased P99 tail latency and reduced total throughput (tokens/sec).

Why Wakeup Latency Matters: The Barrier Tax

- Decode is bulk-synchronous: all workers must rendezvous
- A delayed worker creates idle waiting in the rest of the team
- Small scheduler delays become large throughput losses
- P99 latency is shaped by the slowest thread in each round

The Stutter Seen on Llama Runs

- High CPU utilization but throughput below expectation
- Inter-token gaps showed up in traces despite active cores
- Worker completion times were staggered instead of tightly aligned
- Evidence pointed to placement and wakeup-path distortion

Diagnosis: staggered completion and barrier wait, not lack of compute.

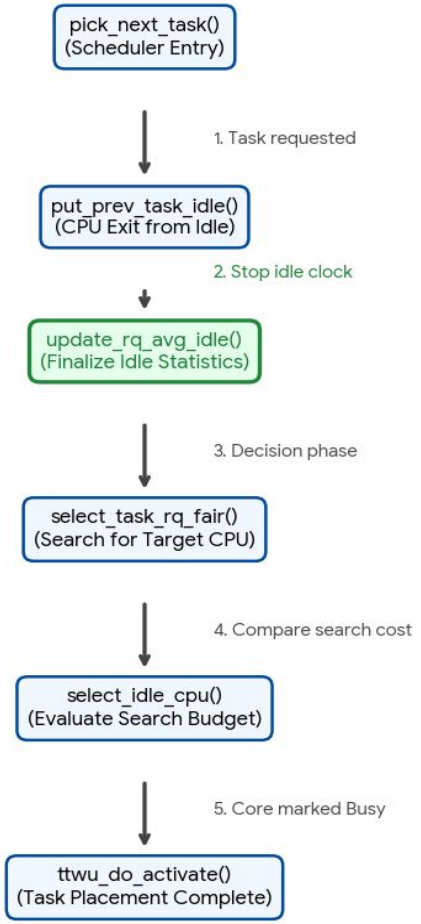
The avg_idle Blind Spot

- avg_idle estimates how long a CPU has really been idle
- The scheduler uses it to decide how deep to search for idle targets
- If the idle clock is stale, a busy CPU can look ‘very idle’
- That creates ghost idle and wrong worker placement

1. Wrong idle telemetry: ghost idle.
2. Ghost idle: wrong CPU choice.
3. Wrong CPU choice: stutter.

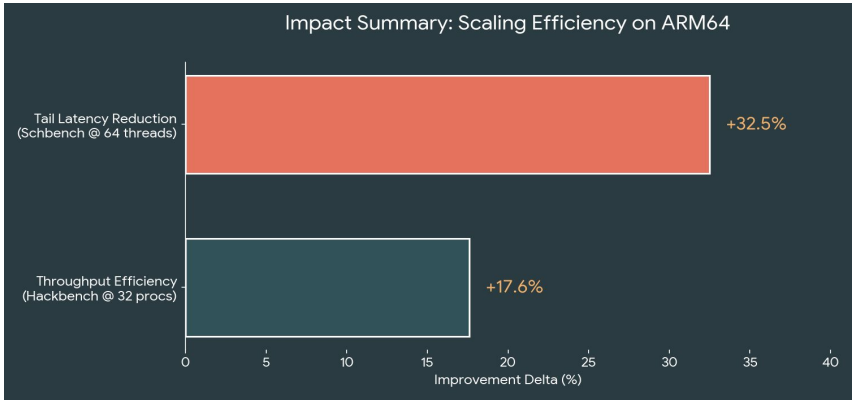
The Patch: Accurate Idle Accounting

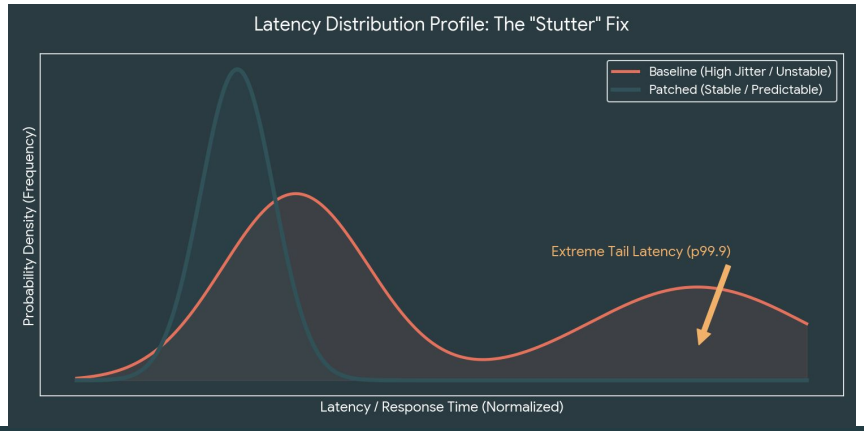
- Stop the idle clock whenever a CPU stops being idle
- Update avg_idle across more task-arrival paths
- Reduce false-idle signals in CPU selection
- Improve placement quality and reduce wakeup jitter



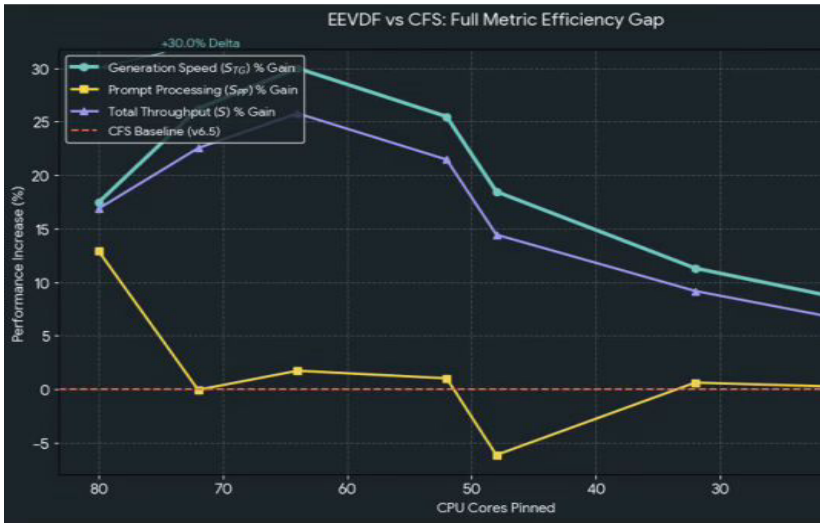
Impact on Normal Scheduler-Sensitive Workloads

- Lower tail latency in latency-sensitive benchmark paths
- Better scaling efficiency when wakeup and distribution dominate
- Cleaner placement decisions at higher core counts
- Less noise from false-idle-driven balancing behavior



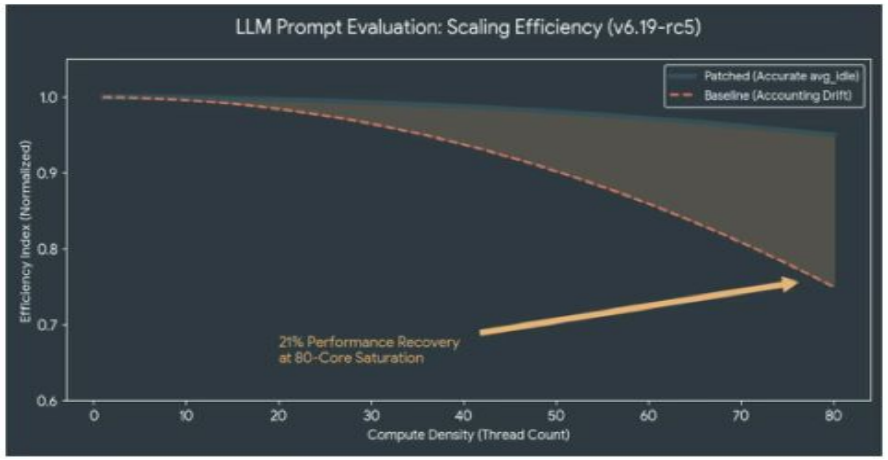


Impact on Llama AI Inference



- Prompt processing benefited from better mass worker placement
- Decode rounds saw tighter worker completion windows
- Lower barrier tax improved throughput and tail behavior
- The system became more predictable at higher core counts

Better idle telemetry leads to better placement, tighter barriers, and higher usable throughput.

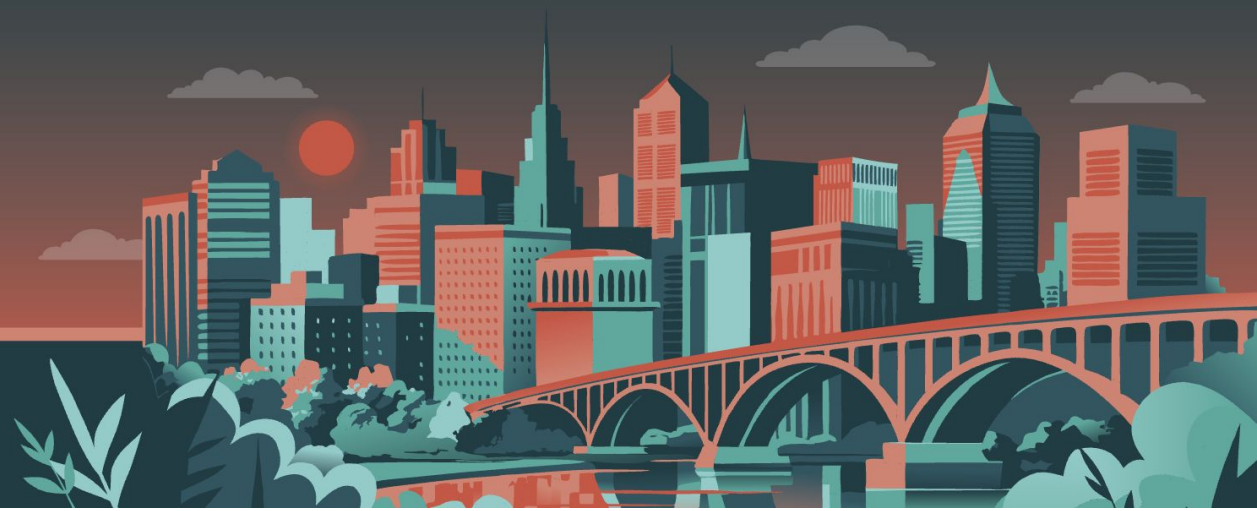


Key Takeaways

- For CPU inference, the scheduler is part of the serving pipeline
- Hackbench, schbench, and cyclictest explain the kernel-side failure modes
- EEVDF is a better fit for bursty sleepers than heuristic sleep credit
- Wakeup latency is shaped by placement quality, not just function cost
- Accurate idle accounting improved both classic and AI workloads
- References: update rq->avg_idle , lwn.net, kernel docs, ampere docs
- AI Images by Nano Banana 2

Thank You! Questions?

LinkedIn : @shubhangpk | shubhangpk@gmail.com



THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA
Embedded Linux Conference

[Additional Info.] AI Stutter mitigation strategies:-

OS Transformation Strategy

- **Goal:** Eliminate "Barrier Tax" and performance stutter.
- **Method**: Change OS from multi-tasker to dedicated engine.
- **Core Principle**: Dedicate Cores - e.g. <1-79> strictly to LLM execution.
- **Housekeeping**: Move all background tasks to hk cpu mask (core 0 for e.g.).

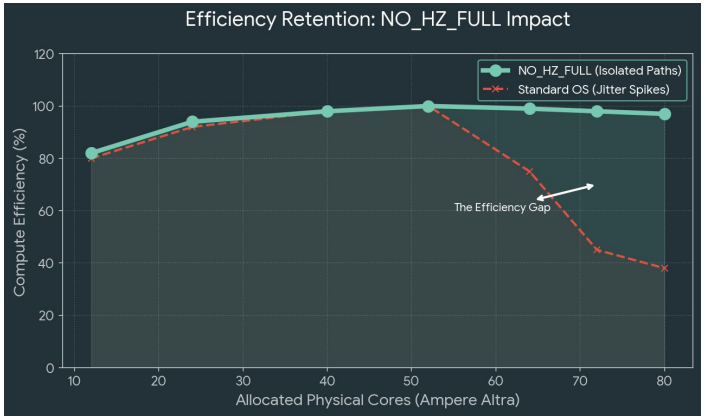
4 Pillars of Low-Latency Tuning

- **CPU Isolation**: isolcpus keeps general work off compute cores.
- **Adaptive-Tick**: Config NO_HZ_FULL stops periodic kernel interrupt pokes.
- **Thread Affinity**: numactl locks inference workers to physical hardware.
- **RCU Offloading**: rcu_nocbs moves memory cleanup to hk cpus.

[Additional Info.] NO_HZ_FULL Impact on AI Inference

Eliminating the Barrier Tax: In standard mode, adding the 80th core increases the chance of an OS interrupt stalling the entire team. In NO_HZ_FULL mode, the 80th core is as quiet as the 1st, allowing the team to scale without friction.

Determinism: The "Gap" in the graph represents the throughput you reclaim simply by silencing the kernel timer tick. On the Ampere Altra, this is the difference between a production-ready AI service and one plagued by "stuttering" response times.



Predictable High-Core Scaling: While the standard OS collapses to lower efficiency at 80 cores, CONFIG_NO_HZ_FULL maintains >95% efficiency.