LAVD: Meta's New Default Scheduler

David Dai & Ryan Newton

Meta

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

- Today:
  - Specialized schedulers exist for targeted services (scx_layered)
  - Substantial performance gains but difficult to tune and maintain
- Goal: A **default** scheduling solution for the rest of the Meta's services.

Meta's scheduling challenges

Varied Hardware: CPU/cache topologies

Varied Workloads: Batch, Bursty, Networking

This talk:

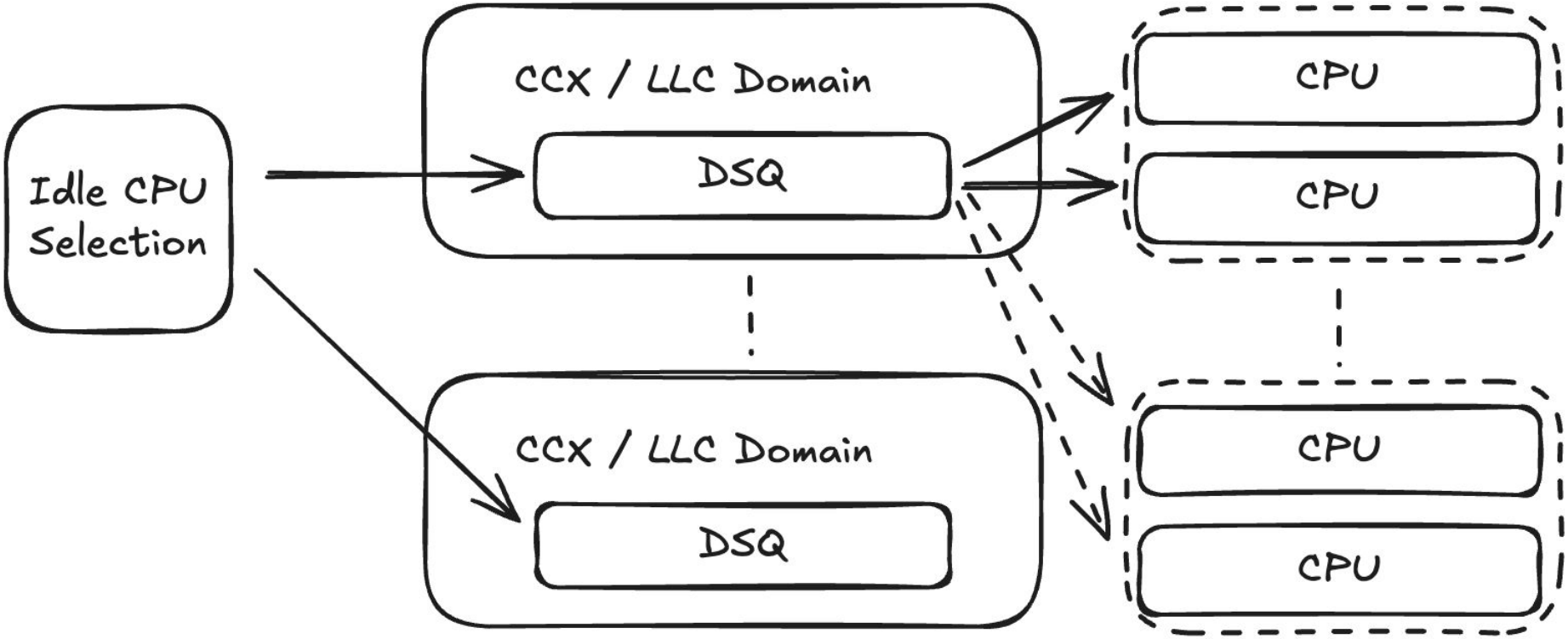
- Problem 1: many CPUs-per-compute complex
- Problem 2: load balancing on large machines
- Problem 3: handling pinned tasks
- Problem 4: IRQ-heavy workloads

Hardware trends

Machines within the fleet are growing in size and in complexity with varied topologies

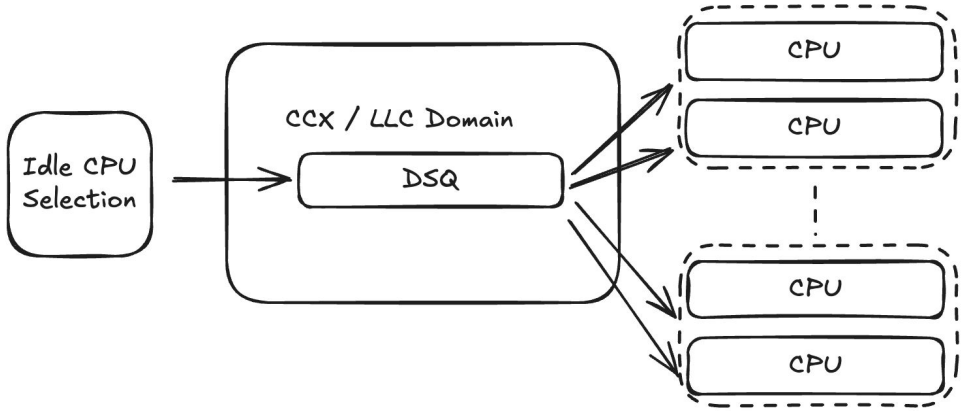
- 10s CPUs -> 100s CPUs
- Single LLC/CCX Domain -> Multiple LLC/CCX Domains
- Multi-socketed machines

LAVD Internals (Simplified)



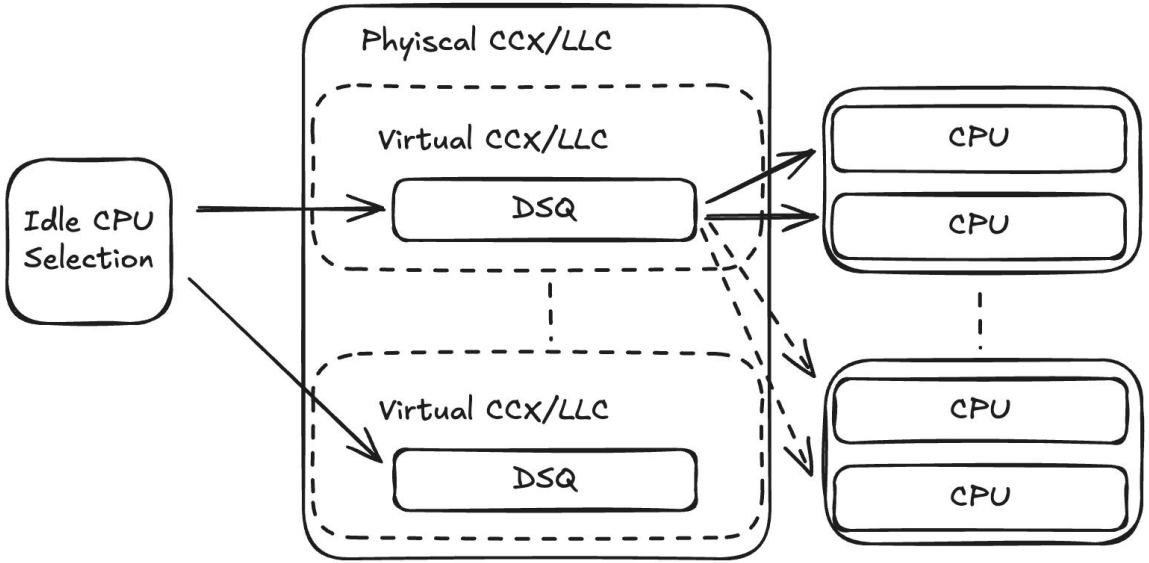
Problem 1: Hardware with a large CCX

- One machine had 50+CPUs with a single CCX domain
- Poor performance due to contention on a single DSQ lock
- Poor L1/L2 Cache hit rates, tasks migrate freely between CPUs



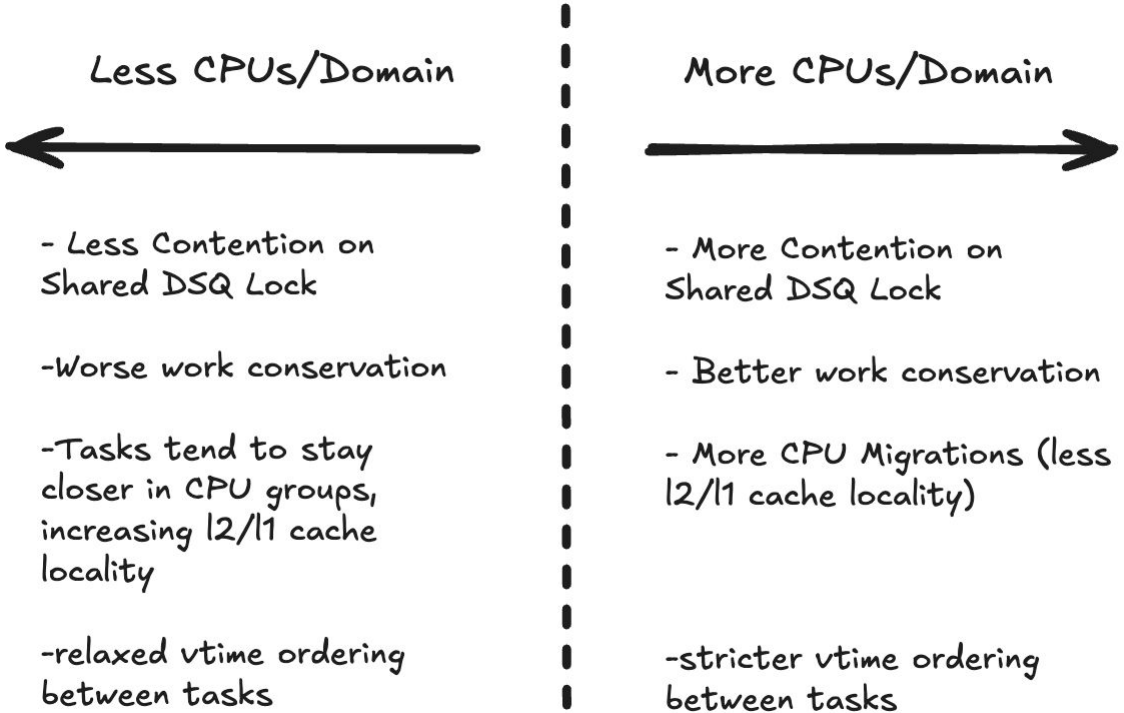
Solution 1: Virtualized CCX/LLC Domains

- What if we subdivided large LLC domains into smaller virtual ones?

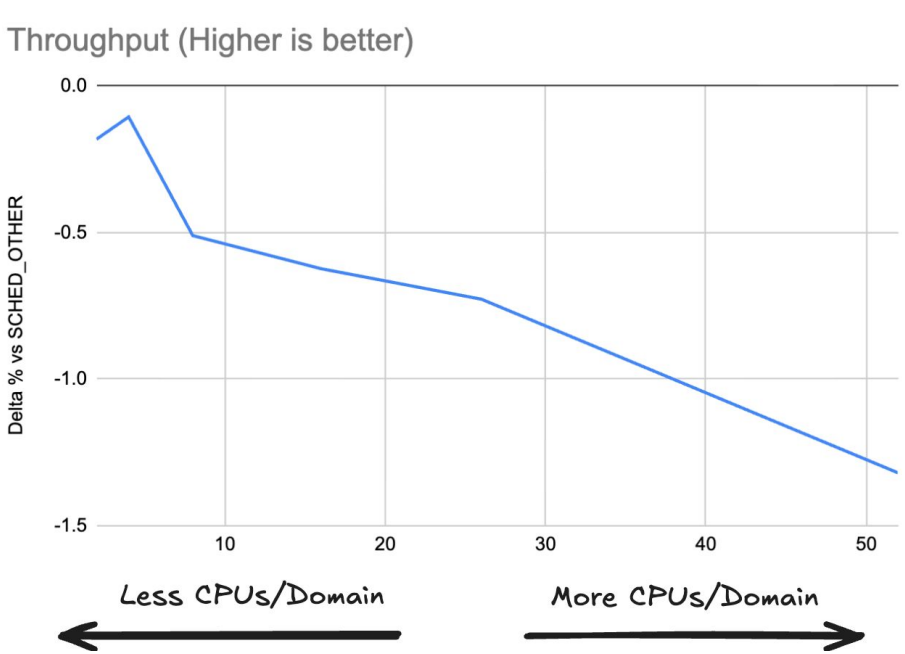


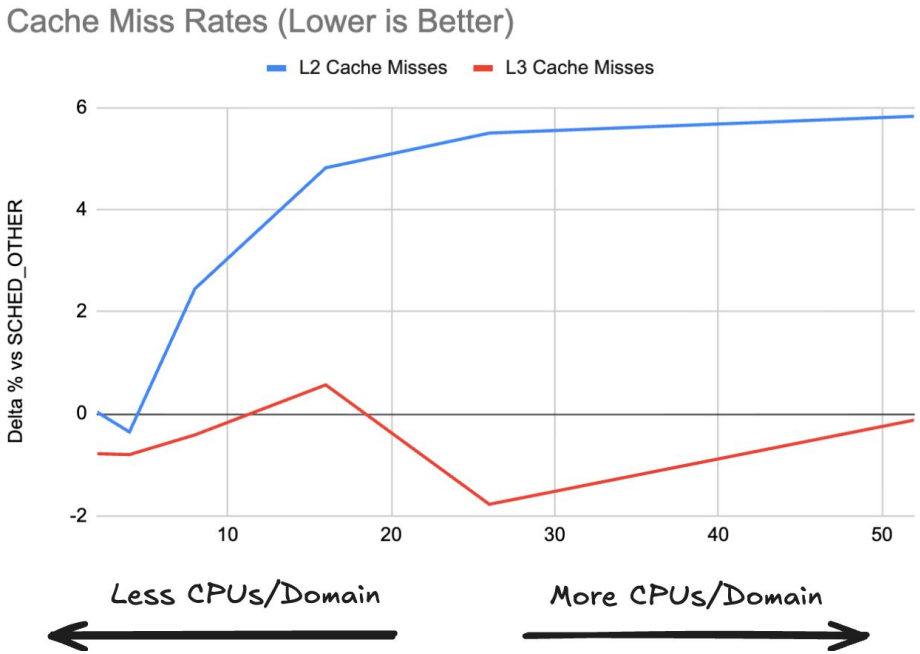
Tradeoffs

- Is there an “optimal” split?



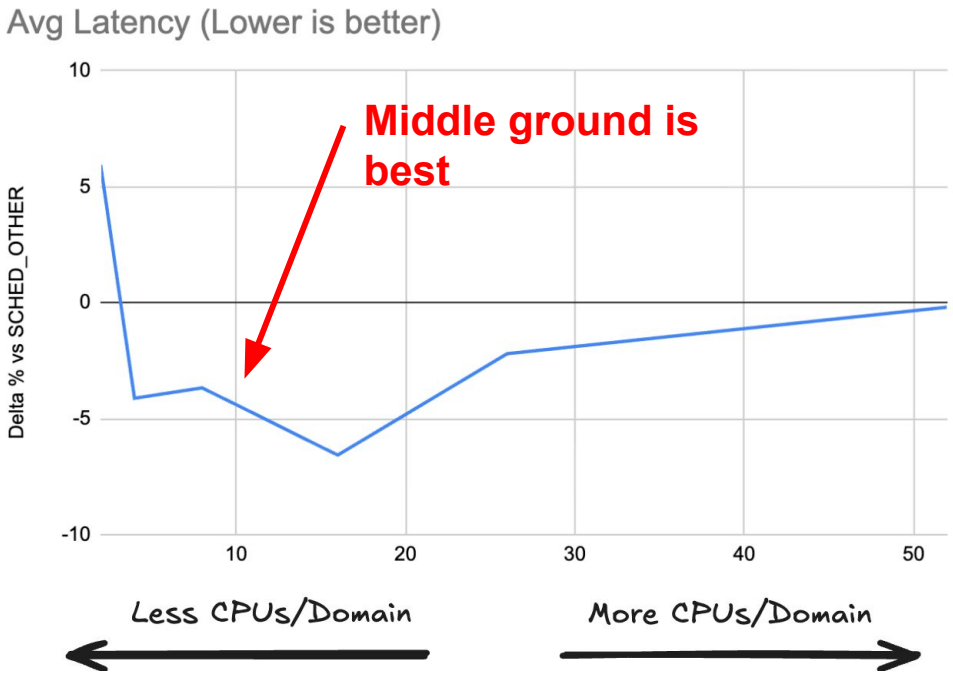
Case Study: A large user-facing service





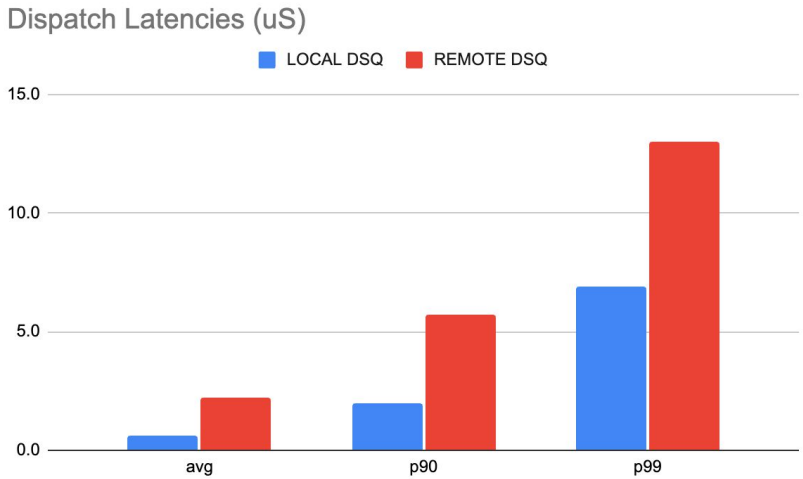
Case Study: A large user-facing service

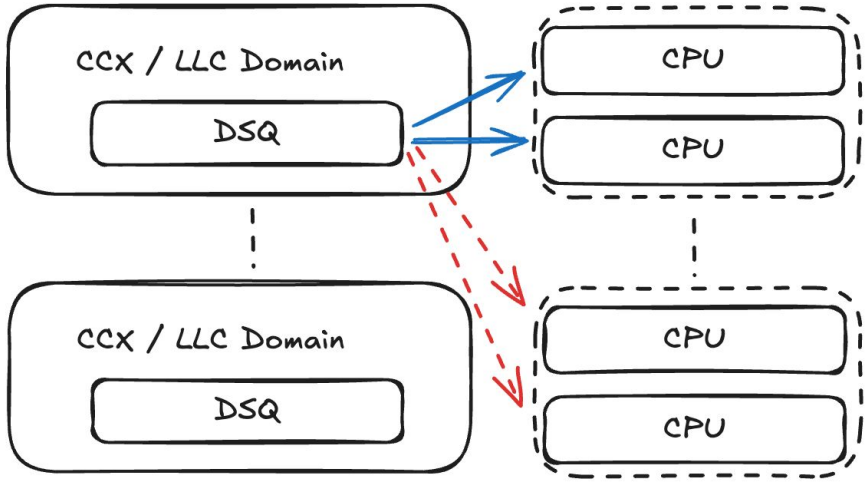
- Workload throughput correlated with L2 cache usage
- At 2 Cpus/Domain, we hit work conservation issues & latency spikes
- A middle ground exists for grouping CPUs in a way that improves cache hits while maintaining work conservation



Problem 2: Load Balancing on Large Machines

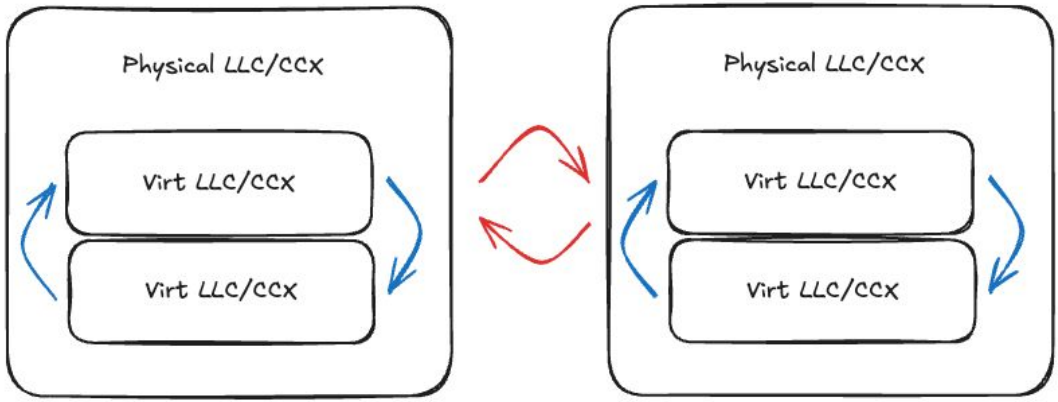
- Stealing across CCX/LLC boundaries is expensive





Solution 2: Tiered Task Stealing

- Discourage task stealing across physical LLC/CCX boundaries
- Favor load balance between physical CCX/LLC on enqueue task
- Tune stealing rate, depending on granularity of Virtual LLC/CCX

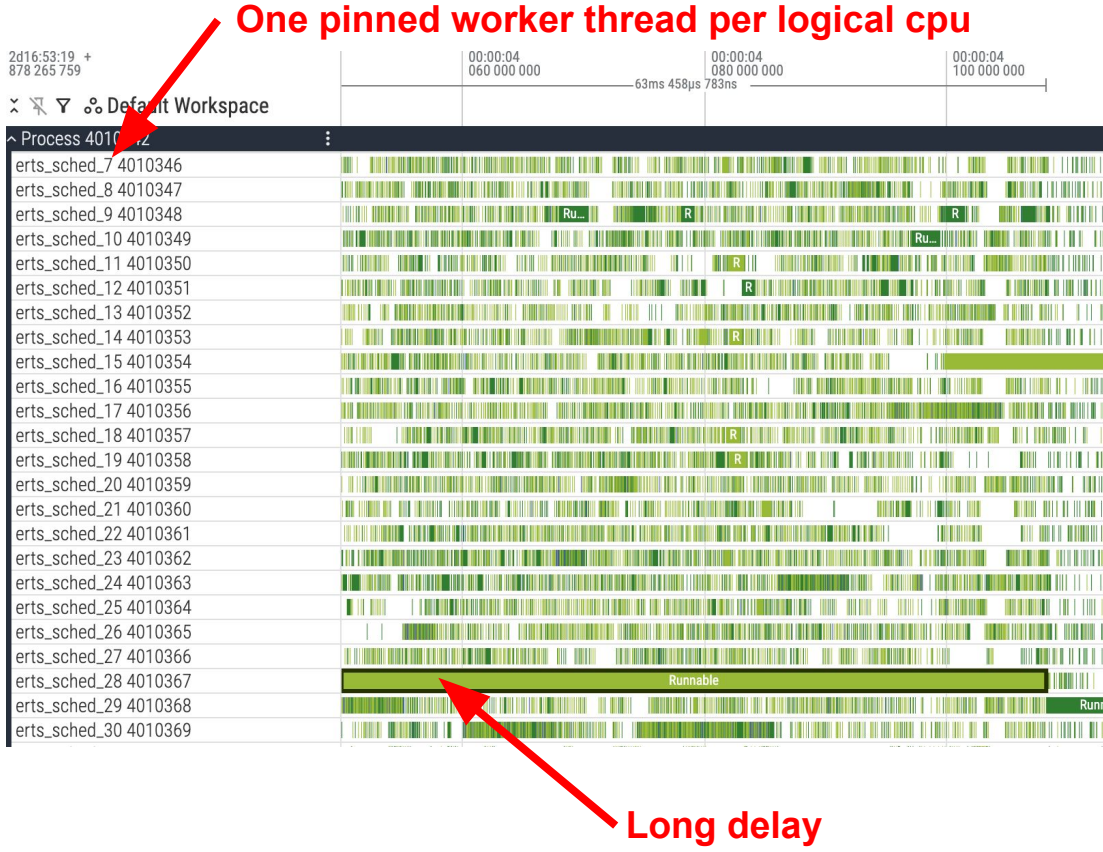


Problem 3: handling pinned tasks

- Pinned tasks would go into LAVD shared DSQ
  - but are only runnable by one core in the CCX!
  - problems ensue...

Problem 3a: P pinned latency-sensitive tasks (Erlang Workloads)

- **erts_sched** threads expecting sub-1ms scheduling delay
- Longer slices exacerbate delay



Problem 3b: LAVD antipattern (forced task stealing)

- Clogged queue ⇒ unpinned tasks more likely stolen by other CCXs

Solution 3: per-cpu DSQs for pinned tasks

- Per-CPU DSQ for all pinned tasks, to prevent contention issues on shared DSQs
  - *Multi-level DSQs — locksfree peek helps*

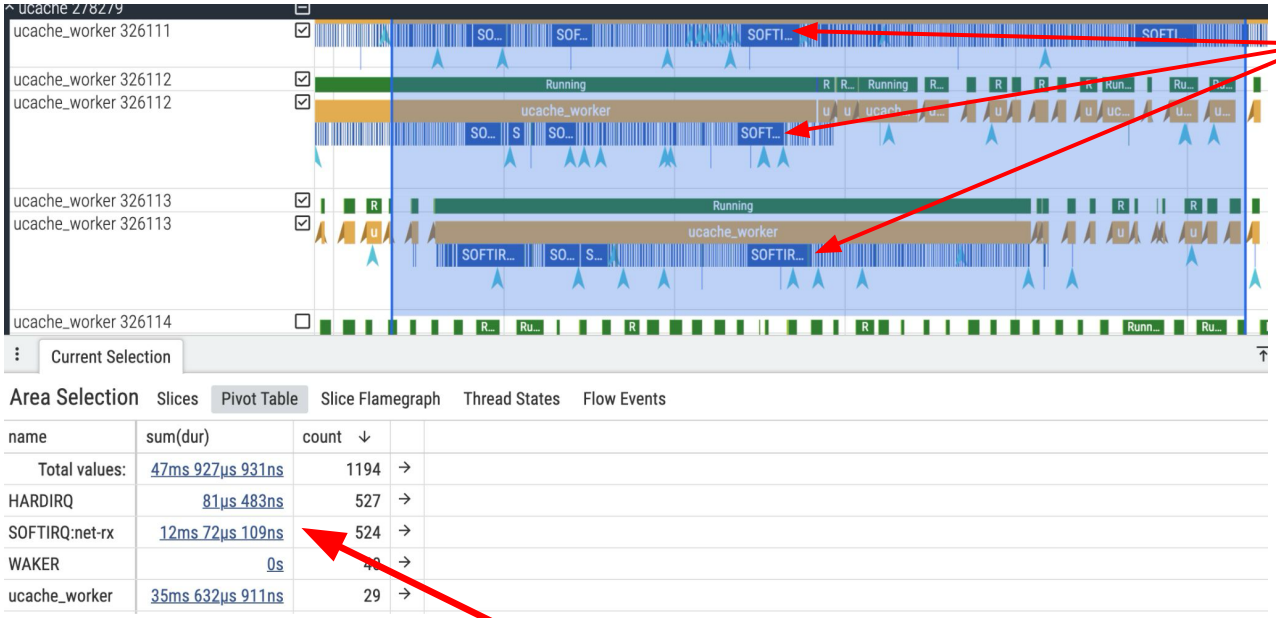
- Dynamically reduce slice times when there are pinned tasks to increase responsiveness (`--pinned-slice-us`)

~~Problem 3~~: Fixed CPU Pinning

Very busy core, multiple pinned tasks

Problem 4: IRQ-heavy workloads

- Here IRQ takes up ~25% of entire system CPU time!



So many SOFTIRQ!

12ms in SOFTIRQ out of 47ms

Solution 4 (wip): irq accounting & balancing tweak

- (1) deduct IRQ time from quota granted to task

- (2) experimenting w/ new load-balancing logic to move latency critical tasks to higher *capacity* cores
  - capacity is decreased by *interrupts*

A topic for another day: benchmarks & reproducers

- Scheduler perf measurement is non-trivial.
  - Noise, isolation problems, lack of standardized benchmark suites.
- We produce one-off reproducers for problems...
  - but we seek to make reproducer-generation process fast and repeatable

This list of problems was non-exhaustive!

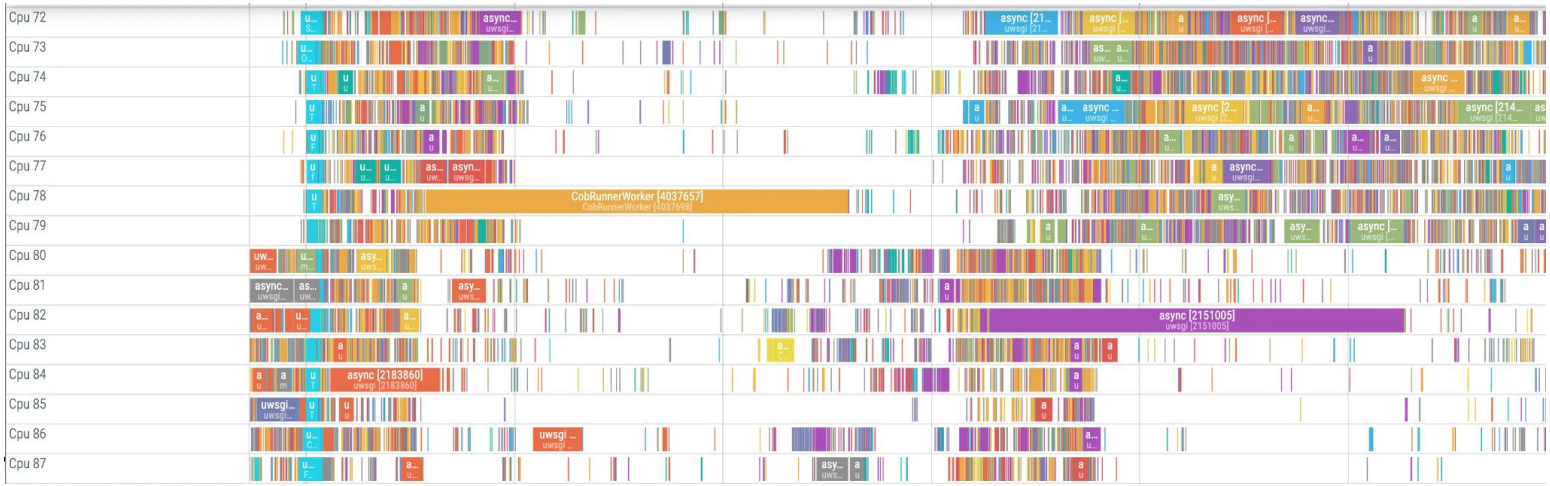
- Wasted “impossible” steals of affinitized tasks between DSQs/CCX domains
- Bursty workloads: Periodic “fork bombs” where workers are killed and restarted on a regular basis
- etc...

BACKUP Slides / Discussion topics

- More Load-Balancing Woes
- Latency Aware Task Placement?
- Push vs Pull?

Load-Balancing Woes:
Average CCX utilization hides skewed distribution

- Smaller tasks are “pushed away” from the same domain as longer running tasks
- LAVD uses summed utilization to load balance
- Poor utilization of L1/L2 caches in CPUs



Latency Aware Task Placement?

- Traditionally both latency and bandwidth requirements are folded into a single signal. (ex. utilization)
- What if we split latency and bandwidth requirements as separate vectors and fit tasks on both dimensions?
- LAVD already tracks which tasks are latency critical or enter critical sections of work:
  - Latency signals are used to calculate virtual deadlines
  - Why not use them to also make task placement decisions?
  - Steer latency critical tasks clear of IRQ heavy CPUs

Push vs Pull?

- CPUs competing to pull tasks naturally self balances and favors idle CPUs
- Idle CPU selection at push doesn't guarantee that it'll run on the suggested CPU
- Load balancing difficult at a domain level.