What the Scheduler Doesn’t Know

Why your inference optimizations aren’t working

Sheetal Sriram
MSCS Student, Georgia Tech SAIL Lab
Intern, NVIDIA BaseOS Foundry

Cloud Native AI + Inference Day / 9 Nov 2026



You’ve tuned the stack

- ❓ KV-cache-aware routing
- ❓ Batch scheduling
- ❓ GPU pod sizing

Some requests still take longer than expected.

The median moved 4%. What about the tail?

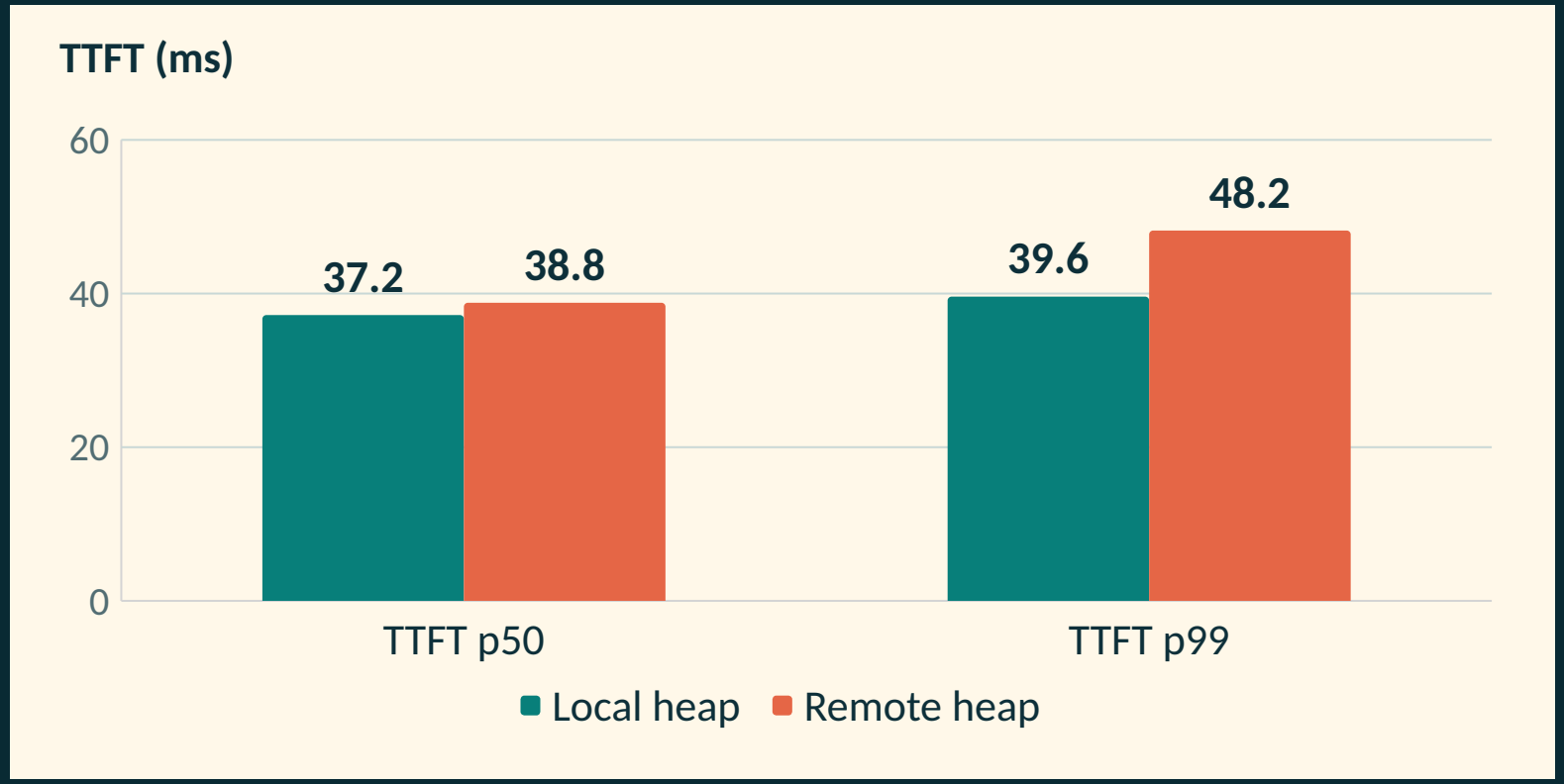
+4%

Median time to first token
with remote host memory

A. ~4%
B. ~10%
C. ~22%

p99 is the latency at the 99th percentile. Local and remote memory-binding arms.

The tail moved 22%



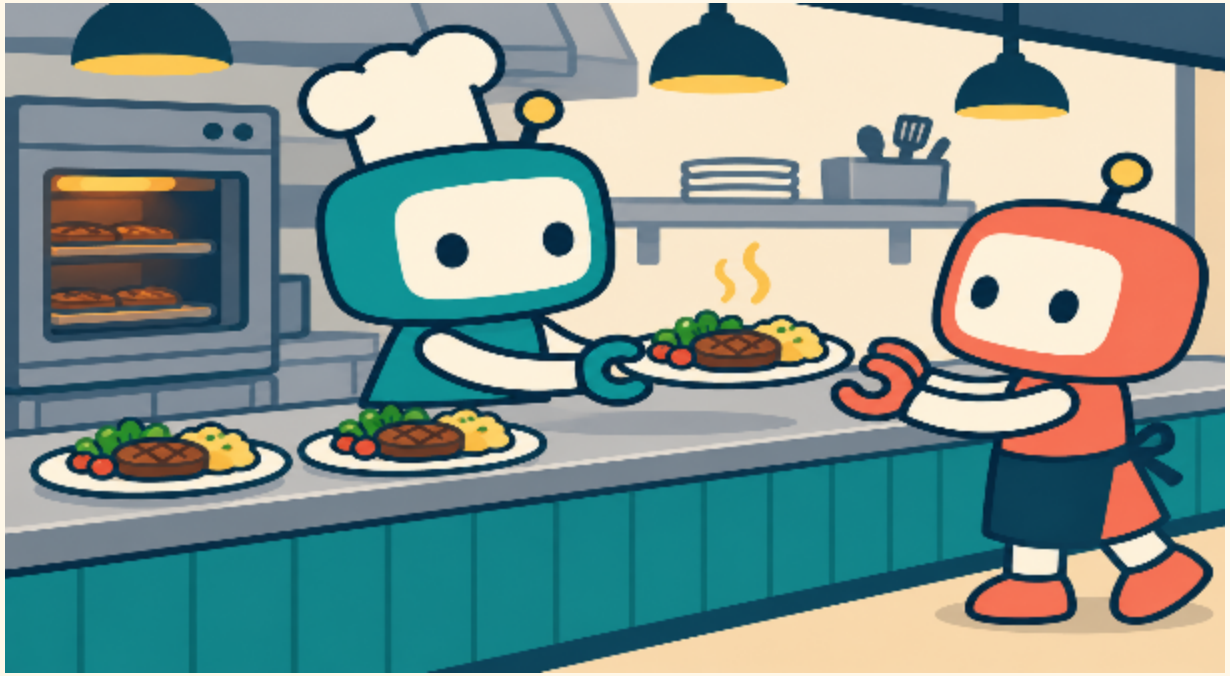
+22%

TTFT p99

C: TTFT p99 rose 22%, while the median rose 4%.

TTFT = time to first token. Two-NUMA-node memory-binding study.

Welcome to the inference kitchen



Request router

Order dispatcher

CPU-side worker

Robot chef

GPU

Oven

Host memory

Pantry

Latency and throughput

Latency

Time to complete
one order

For inference: milliseconds / request

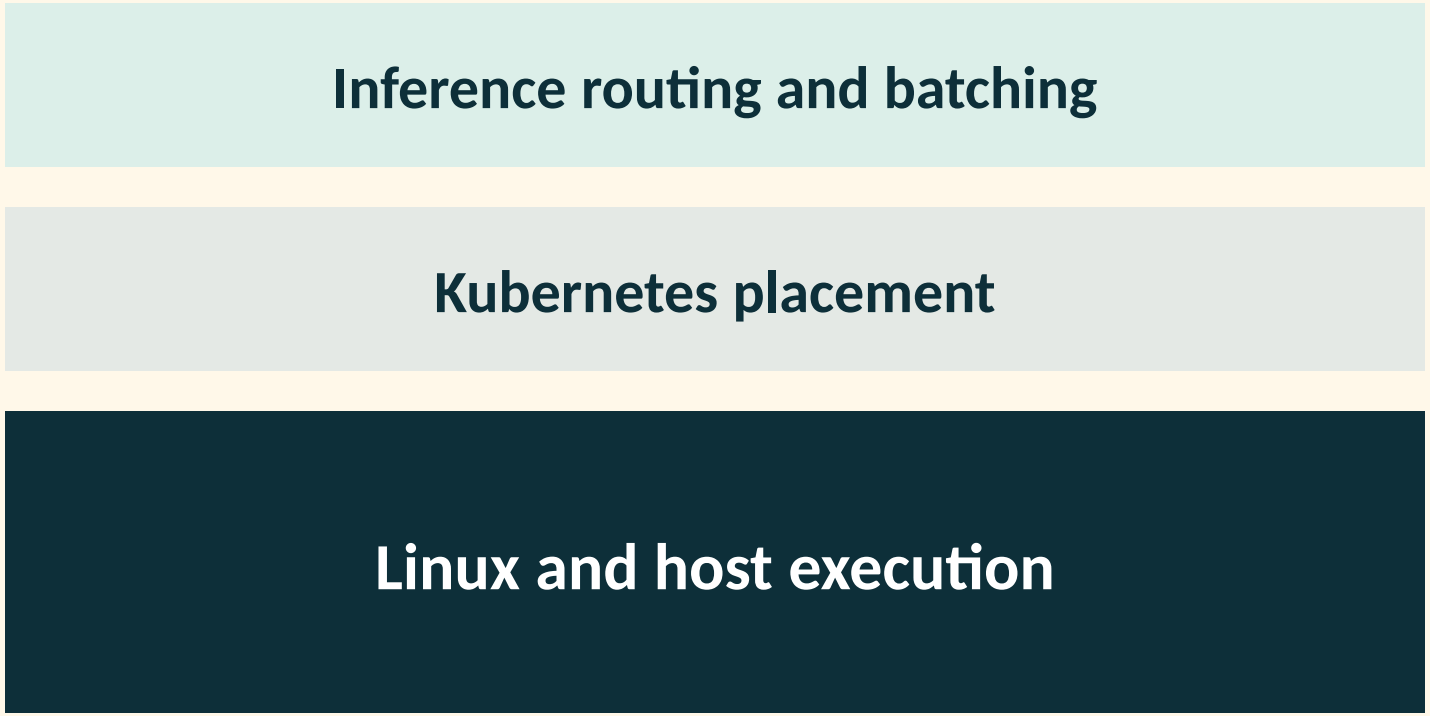
Throughput

Completed orders
per second

For inference: requests/s or tokens/s

What the serving dashboard doesn’t explain

Left: system layers. Right: what to observe at each layer.



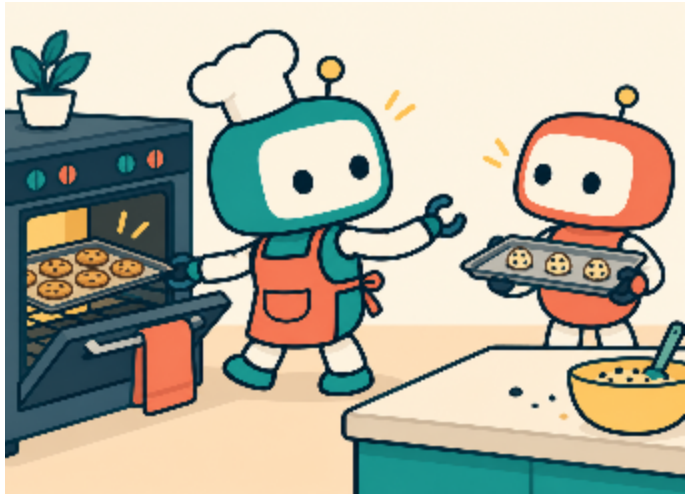
KV-cache state
Endpoint load

Requests, limits
Topology hints

Runnable delay
Heap locality
IRQ affinity and activity

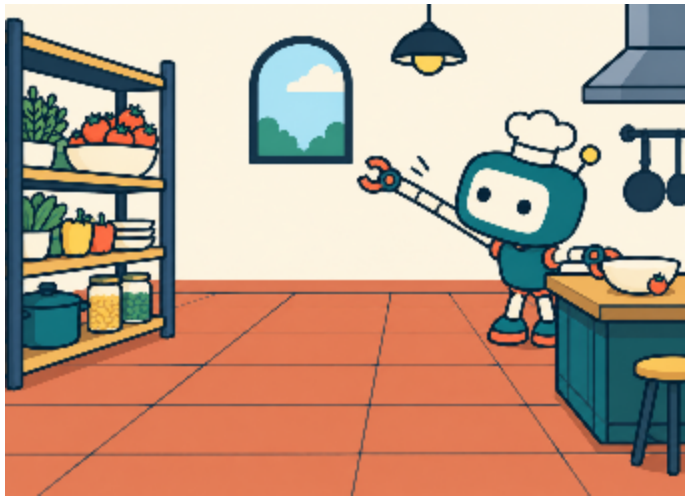
Host layer: execution and locality

Three things that can keep the oven waiting



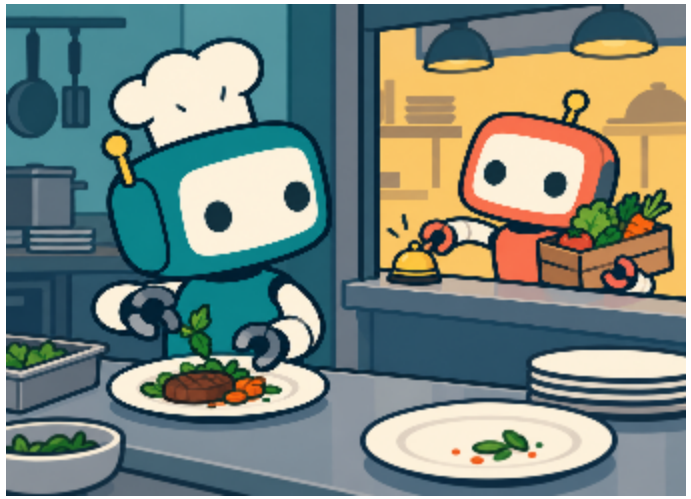
CPU scheduling

The chef waits for CPU time



NUMA locality

The pantry is farther away

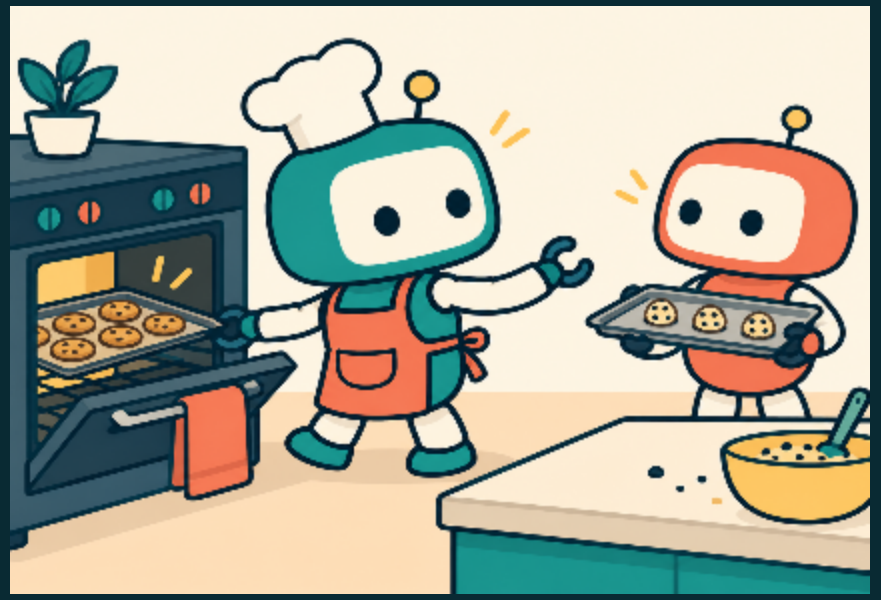


IRQ affinity

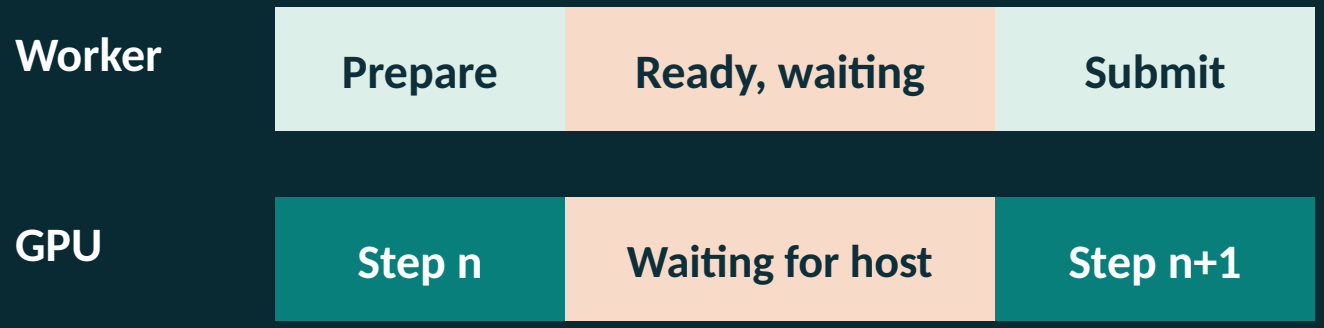
Deliveries can interrupt the chef

CPU time, memory placement and interrupts need different checks.

The chef is ready, but waiting for a turn



CPU scheduling



Illustrative timing, not a measured trace

Runnable = ready, waiting for CPU time

Does a CPU request guarantee dedicated CPU time?

A CPU request alone doesn’t give exclusive CPUs

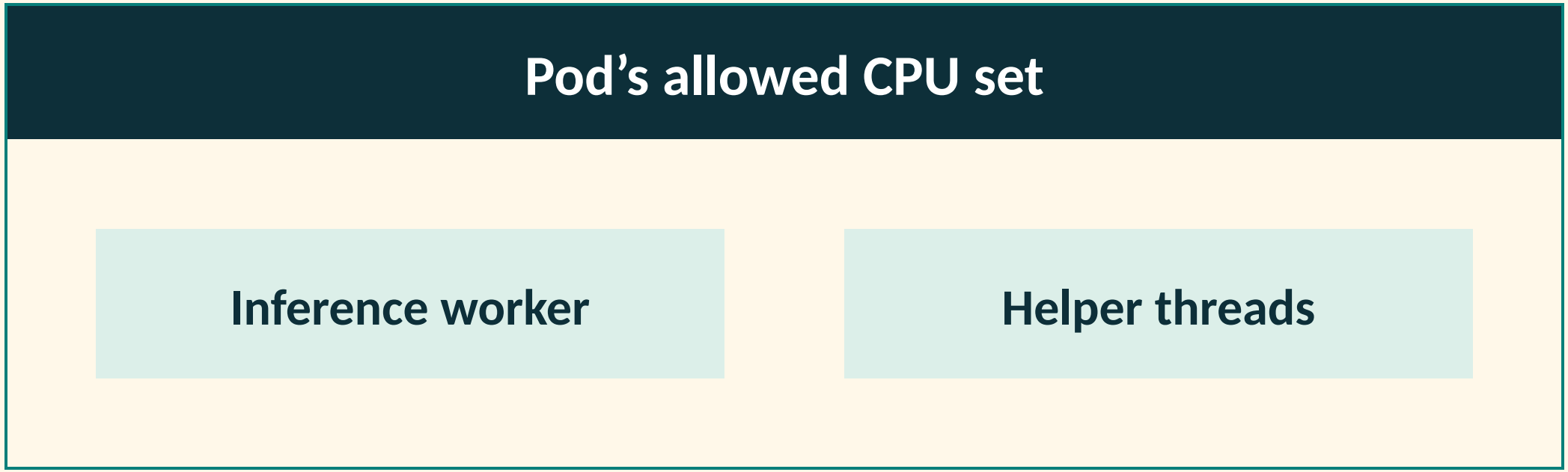
Kubelet policy

cpuManagerPolicy: static

Eligible pod: Guaranteed QoS + integer CPU request

Guaranteed QoS: CPU and memory requests equal limits for every container.

The engine must honor its allowed CPU set

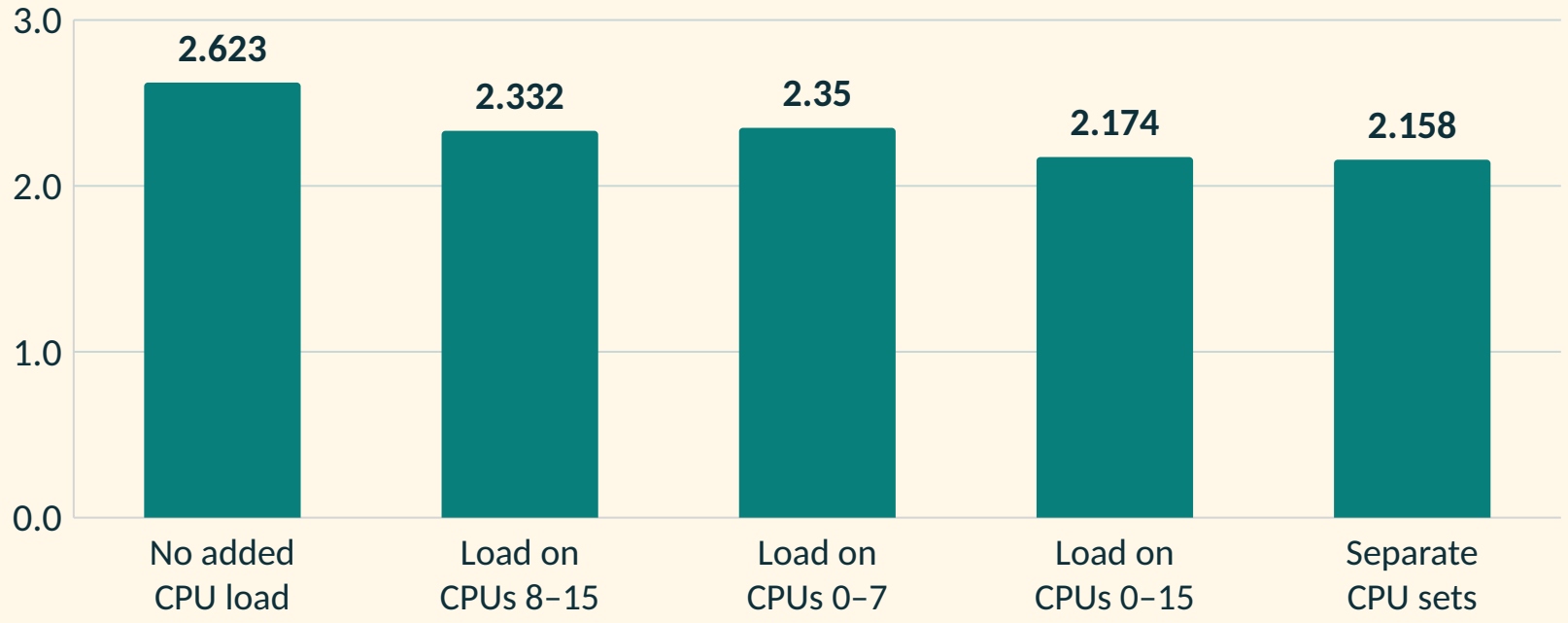


Keep helper threads within the pod’s allowed CPU set.

Inspect per-thread affinity: /proc/<PID>/task/<TID>/status

Similar GPU averages, different request tails

End-to-end p99 latency (seconds), all requests

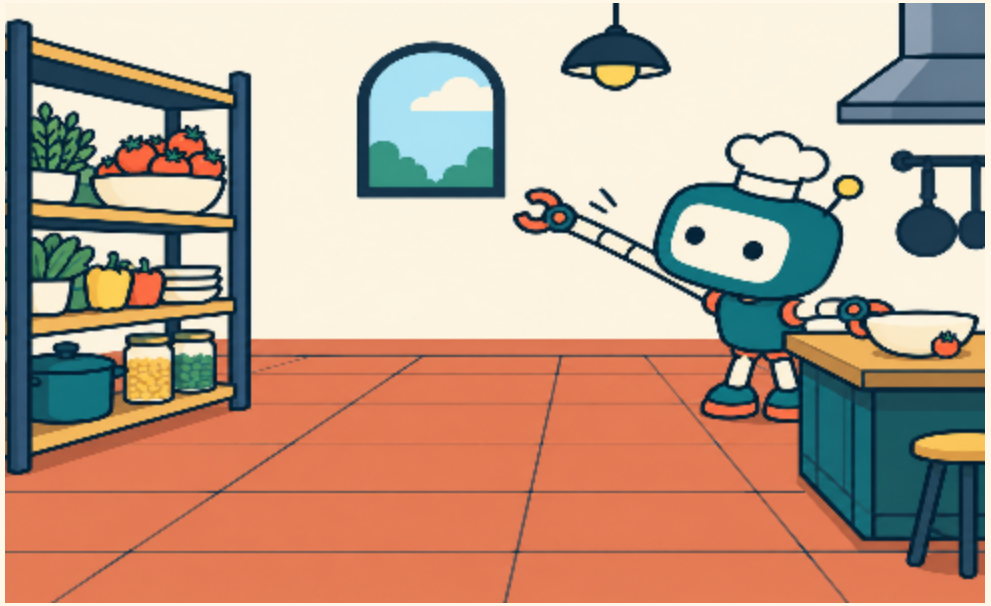


58–64%

Range across all five runs
Mean GPU SM utilization
60 one-second samples/run

First four runs: serving CPUs 0–15. Last: serving 0–7, load 8–15.
Placement and competing load both vary. The comparison does not isolate pinning.

Same chef. Same oven. A farther pantry.



NUMA 0

Remote arm
~464 MB heap

NUMA 1

CPUs 56–63 + GPU

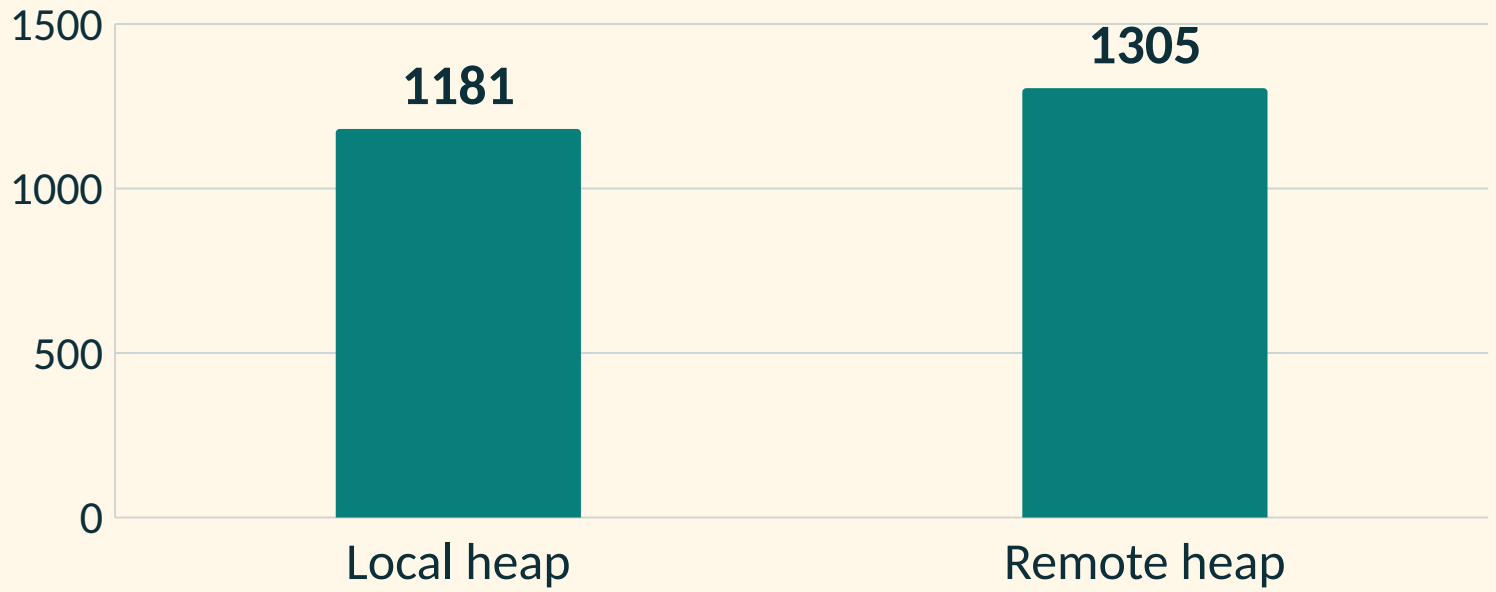
Local arm
~448 MB heap

CPU binding stayed fixed. Only host-memory binding changed.

NUMA = non-uniform memory access. Host heap is separate from GPU-resident KV tensors.

Remote memory also raised complete-request tails

End-to-end p99 latency (ms)



~+10%

Same memory-binding comparison. Source: NUMA study summary.

The chef, pantry and oven need to line up

```
cpuManagerPolicy: static
topologyManagerPolicy: single-numa-node
memoryManagerPolicy: Static
```

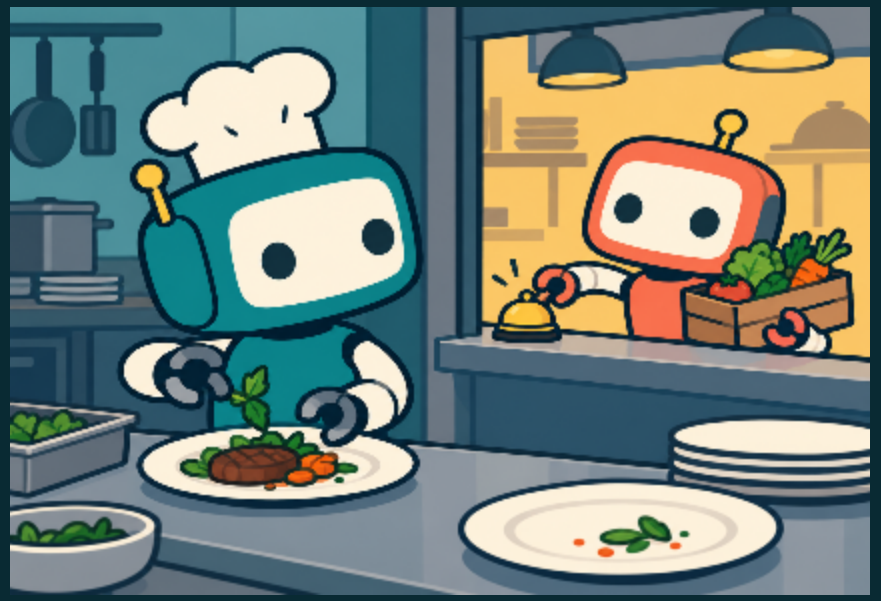
Compatible CPU, memory and GPU placement

CPU | **Host memory** | **GPU**

Can a NIC still interrupt a CPU inside the inference pod's allowed set?

Requires Guaranteed QoS, device topology hints and configured memory reservations.

Yes. Interrupt affinity is a separate setting



NIC completion interrupt (IRQ)

Example inference CPU set

56	57	58	59	60	61	62	63

Check effective affinity to identify target CPUs.

A pod CPU set does not exclude hardware interrupts.

Illustrative CPU set. The shown IRQ capture has no listed increases on CPUs 56–63.

Deliveries need a place to land



smp_affinity_list

/proc/irq/<IRQ>/smp_affinity_list

Verify the effective affinity, then repeat the workload.

Inference CPU set

Eligible NIC IRQs can run on CPUs outside the inference set.

Host privileges required. Coordinate with irqbalance and driver constraints.

Following the workload into the host

Host profiling walkthrough

GPU	CPU	Memory	IRQs
Activity gaps	Runnable delay	Page residency	IRQ routing

Observe the workload. Change one variable. Repeat the measurement.

Three observations to take back to your cluster

CPU time	Verify exclusive CPU allocation and each thread’s effective affinity.
Memory locality	+22% TTFT p99 with remote host heap
Interrupts	Measure IRQ affinity and activity separately

vLLM: the pod’s CPU boundary matters

Open PR #58036: pending merge

CPU NixlConnector can reserve host CPUs
outside the container’s allowed set.

**Next: review the affinity fix and run
regressions inside real cgroup CPU limits.**

CPU-backend startup failure. Related evidence for affinity correctness, not a GPU latency result.

#58033: report

#58036: proposed fix

Status as of 4 Nov 2026

NVIDIA Dynamo: host visibility is shipping

Shipped in v1.6.0

NUMA affinity warning at frontend startup

Sustained Tokio runnable-queue warnings

Next: effective worker placement diagnostics and constrained-container regression tests.

These features independently recognize host placement and contention. This talk measures vLLM.

v1.6.0 release notes: #11184 and #13027

Release notes verified 4 Nov 2026

Kueue and CPU DRA: placement foundations

Available today

Kueue topology-aware scheduling: beta since v0.14

Kubernetes CPU DRA driver v0.3.0
Exclusive CPU allocation

Next: validate admission, CPU accounting and device alignment together.

KEP-5517: alpha since 1.36. Beta target 1.38, off under default compatibility.
Default-on target 1.39. This is node resource accounting.

Kueue TAS kubernetes-sigs/dra-driver-cpu KEP-5517

Kueue and KEP status checked 4 Nov 2026

llm-d and Gateway API: testing host signals

The routing extension points already exist

Gateway API Inference Extension v1.5.0
includes metrics data-source plugins.

Proposed: validate NUMA locality and CPU wait as inputs to endpoint selection.

Compare with the same routing policy without host signals. Measure tails, stale-data behavior and overhead.

v1.5.0 metrics data-source API

Status checked 27 Oct 2026

The next gains are one layer down

The control plane can’t optimize what it can’t see.

Open Q&A

https://www.linkedin.com/in/sheetals01

https://github.com/sheetal104/what-the-scheduler-doesnt-know

CPU placement study

Qwen3-0.6B / vLLM 0.31 / --enforce-eager

Setup	Load
16 vCPU, 1 NUMA node, 1 GPU	512 input / 128 output tokens
Five placement configurations	c=16, 300 s per arm
40 warmup requests	2,400–2,528 requests per arm

Source: CPU placement study configuration and request logs.

NUMA memory-binding study

Fixed across arms	Memory binding
GPU associated with NUMA node 1	Local arm: node 1
vLLM CPUs 56–63	Remote arm: node 0

512 input / 128 output, c=8, 120 s per arm

832 local-arm requests / 800 remote-arm requests

Qwen3-0.6B / vLLM 0.31 / eager execution. Source: NUMA study summary.

Read-only NIC interrupt capture

vLLM CPUs 56–63

Qwen3-0.6B / c=8 / 90 s / 640 requests

TTFT p50 36.5 ms / p99 40.1 ms

E2E p99 1,160 ms

Source: IRQ study summary. IRQ affinity was not changed in this capture.

GPU activity and topology

```
nvidia-smi dmon -s u -d 1

nvidia-smi topo -m
```

dmon samples GPU activity. topo shows device and CPU topology.

CPU scheduler delay

```
perf sched record -a -- sleep 30
perf sched latency
perf sched timehist -p <PID>
sudo runqlat-bpfcc -p <PID> 1
```

Tracing may need host privileges. BCC binary names vary by distribution.

Host-memory residency and CPU mask

```
numactl --hardware
numastat -p <PID>
grep Cpus_allowed_list /proc/<PID>/status
grep Cpus_allowed_list /proc/<PID>/task/<TID>/status
```

Use the serving worker PID. Residency and allowed CPUs answer different questions.

NIC interrupt counters and affinity

```
awk 'NR==1 || /mlx/' /proc/interrupts > t0.txt
# Repeat as t1.txt during the workload
diff -u t0.txt t1.txt
cat /proc/irq/<IRQ>/smp_affinity_list
cat /proc/irq/<IRQ>/effective_affinity_list
```

The CPU header identifies which counter changed. Affinity writes require host privileges.

NUMA evidence — vLLM process memory (MB)

membind_local PID 2354592

numastat -p, end of run

	Node 0	Node 1	Total
Huge	0.00	0.00	0.00
Heap	**0.00**	**448.46**	**448.46**
Stack	0.00	0.09	0.09
Private	19.94	875.76	895.70
Total	**19.94**	**1324.31**	**1344.25**

membind_remote PID 2418523

numastat -p, end of run

	Node 0	Node 1	Total
Huge	0.00	0.00	0.00
Heap	**463.95**	**0.00**	**463.95**
Stack	0.08	0.00	0.08
Private	567.29	328.59	895.88
Total	**1031.32**	**328.59**	**1359.91**

CPUs 56–63 on node 1, one GPU on node 1. Local: heap and private memory on node 1. Remote: heap on node 0.

Heap placement changed; CPUs stayed on 56–63.

IRQ evidence — mlx interrupts that increased during the 90 s load

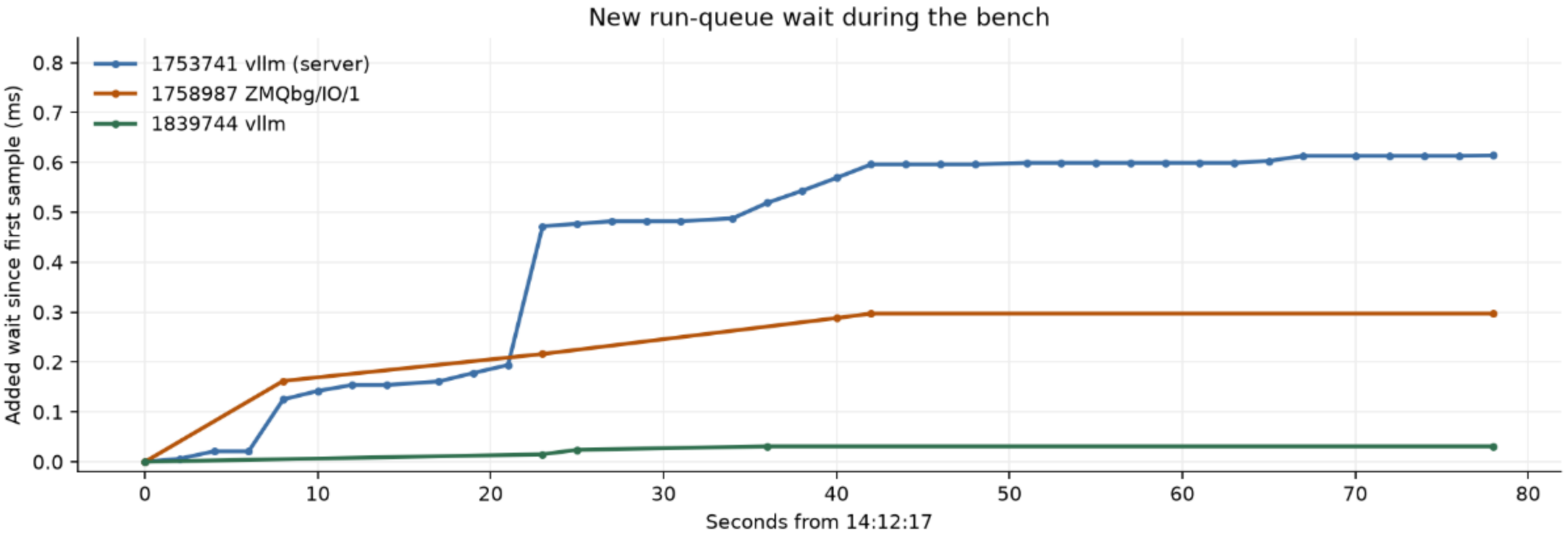
vLLM CPUs 56–63 on node 1. None of these IRQs incremented on that set.

Queue	Device	CPU	Count	Node
mlx5_comp0	**0000:05:00.0**	**CPU0**	**+63**	**node 0**
mlx5_async0	0000:05:00.0	CPU126	+51	node 0
mlx5_async0	0000:05:00.1	CPU127	+51	node 0
mlx5_async0	0000:05:00.2	CPU128	+51	node 0
mlx5_async0	0000:05:00.3	CPU129	+51	node 0
mlx5_async0	0000:18:00.0	CPU130	+51	node 0
mlx5_async0	0000:29:00.0	CPU131	+52	node 0
mlx5_async0	0000:29:00.1	CPU147	+20	node 0
mlx5_async0	0000:40:00.0	CPU156	+51	node 0
mlx5_async0	0000:4f:00.0	CPU157	+51	node 0
mlx5_async0	0000:5e:00.0	CPU158	+51	node 0
mlx5_async0	0000:9a:00.0	CPU175	+51	node 1
mlx5_async0	0000:aa:00.0	CPU176	+52	node 1
mlx5_async0	0000:aa:00.1	CPU177	+20	node 1
mlx5_async0	0000:c0:00.0	CPU178	+51	node 1
mlx5_async0	0000:ce:00.0	CPU179	+51	node 1
mlx5_async0	0000:dc:00.0	CPU180	+51	node 1

1024 mlx lines, 224 CPUs. 17 lines increased. The only completion queue is mlx5_comp0 on CPU 0. The rest are mlx5_async, about +51 each.

Listed IRQ increases lie outside CPUs 56–63.

Scheduler evidence — vLLM threads on CPUs 56–63



31 threads sampled every 2 s. CUDA and most vLLM threads stayed flat. The server thread already had 29.1 ms of lifetime wait; it added 0.61 ms during this window.

Pinned baseline: Cpus_allowed_list 56–63; 31 threads; ~78 s.

Server thread: 29.10 ms already accumulated, then +0.61 ms during sampling.

Controls at a glance

Concern	Control
CPU allocation	cpuManagerPolicy: static
NUMA placement	topologyManagerPolicy: single-numa-node memoryManagerPolicy: Static
NIC IRQ routing	/proc/irq/<IRQ>/smp_affinity_list

Pod eligibility, topology hints and reservations apply. IRQ control requires host privileges.

Comparable experiments

Keep fixed	Everything except the setting under test
Repeat	Matched arms on the same deployment
Report	Warmup, sample count and percentile method

Latency percentiles need an explicit method

CPU placement results

All recorded requests
Interpolation at (n−1) × 0.99

CPU study summary

Trim to the 1st–99th percentiles, then compute p99

All-request p99 keeps the slow requests. Trimming first changes the tail being measured.

CPU p99 with and without trimming

Arm	Trimmed E2E p99	All-request E2E p99
Isolated	2,564.9 ms	2,623.5 ms
Shared neighbor	2,257.1 ms	2,332.2 ms
Overlap scenario	2,189.3 ms	2,349.6 ms
Oversubscribed	2,104.3 ms	2,174.3 ms
Pinned production	2,119.9 ms	2,157.9 ms

Source: CPU study summary and recalculation from per-request logs.

CPU placement arms and request counts

Arm name	vLLM CPU mask	Synthetic hog mask	Requests
Isolated	0–15	None	2,400
Shared neighbor	All / unpinned	8–15	2,403
Overlap scenario	All / unpinned	0–7	2,494
Oversubscribed	All / unpinned	0–15	2,483
Pinned production	0–7	8–15	2,528

Host and workload shape the result

Model and token mix

Concurrent CPU load

CPU and device topology

Validate placement changes with matched runs on the target deployment.

Sources: experiments and profiling

Talk repository

https://github.com/sheetal104/what-the-scheduler-doesnt-know

perf sched

https://man7.org/linux/man-pages/man1/perf-sched.1.html

BCC runqlat

https://github.com/iovisor/bcc/blob/master/tools/runqlat_example.txt

NVIDIA System Management Interface

https://docs.nvidia.com/deploy/nvidia-smi/index.html

Sources: resource placement and affinity

Kubernetes Topology Manager

https://kubernetes.io/docs/tasks/administer-cluster/topology-manager/

Linux IRQ affinity

https://docs.kernel.org/core-api/irq/irq-affinity.html

vLLM constrained-container CPU affinity

https://github.com/vllm-project/vllm/issues/58033

DRA node resource accounting (KEP-5517)

https://github.com/kubernetes/enhancements/issues/5517

Characterizing CPU-Induced Slowdowns in Multi-GPU LLM Inference

https://arxiv.org/abs/2603.22774

Dynamo: placement fixes already released

v1.1.0

PinnedAllocator receives the worker’s device ID
instead of hardcoding GPU 0.

NUMA binding respects CUDA_VISIBLE_DEVICES.

These are specific mapping fixes. They do not establish that all NUMA placement problems are resolved.

#6809 #6931 v1.1.0 release notes

Release evidence checked 27 Oct 2026

CPU DRA driver: CPU claim accounting

Issue #6 closed through merged PR #50

Sharing one CPU claim could produce inconsistent scheduler and driver accounting.

The fix rejects claim sharing and adds tests.

Merged 9 Feb 2026. The first release containing this fix was not confirmed. KEP-5517 tracks the broader node resource accounting model.

dra-driver-cpu #6

#50: merged fix

KEP-5517

Status checked 27 Oct 2026

Fleet efficiency needs a useful-token metric

Output tokens from requests meeting the SLO

per GPU-hour

More useful tokens per GPU-hour can reduce capacity needed for a fixed workload.

Keep model quality, traffic mix and latency SLO fixed.
This talk measures latency effects, not fleet cost savings.

If this happened to you, that’s the starting point

Check numastat and /proc/interrupts on your cluster.
If you find something, share the repro.

Check open issues in vLLM, dra-driver-cpu and Gateway API IE.
If your repro matches, add what you found.

If you maintain one of these projects, track OS-layer correctness through releases.