Episode Book

EPISODE #6552

How Slurm and Kubernetes Split GPU Workloads

PUBLISHED	13 November 2026 at 16:45 UTC
DURATION	32:24
LISTEN	myweirdprompts.com/episode/gpu-cluster-scheduling-slurm-kubernetes-503303/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.26
SOURCES	22 cited by the research (listed at the end)
BOOK GENERATED	14 November 2026 at 06:47 UTC

SYNOPSIS

Standing up a GPU cluster splits into two stacked problems: scheduling jobs onto hardware and keeping collective communication fast. Distributed training stalls if even one GPU is missing, yet Kubernetes's default scheduler places pods one at a time. This walkthrough compares Slurm's atomic batch queues with Kubernetes tools like the NVIDIA GPU Operator, Kueue, Volcano, KubeRay, and Kubernetes 1.37's gang scheduling—explaining why national labs and 66% of organizations running generative models split training and inference across both systems.

TRANSCRIPT



Corn

Eight GPUs requested, six free. Guess what the default Kubernetes scheduler does.



Herman

Starts six of them and leaves two pods pending, every time. Which is a lovely way to open, because Daniel's prompt this week is basically asking for the episode he wishes existed before anyone buys their first rack of GPUs.



Corn

He wants the foundational walkthrough. If you're standing up your first GPU farm, what do you need to know — the scheduling and orchestration side, so Slurm, Kubernetes with the NVIDIA GPU Operator, Ray, and then the communication libraries living underneath all of that, NCCL and its relatives. And then the two scenarios everyone actually hits: can you mix AMD and NVIDIA in one cluster at all, and if you stay with one vendor, what happens when your cluster is a museum of different GPU generations.



Herman

The first thing to internalize is that this is not one problem. It's two, stacked.



Corn

Two problems. Separate failure modes. My favourite kind of architecture.



Herman

The scheduler decides which GPUs a job gets and when it gets them. The collective library decides how fast those GPUs can talk to each other once they're assigned. And here's the trap: a cluster can have perfect scheduling and terrible throughput, because the communication layer quietly fell back to TCP over Ethernet instead of RDMA over InfiniBand. Nobody tells you. It just runs slow.



Corn

And the reverse.



Herman

You can have a beautifully wired NVLink domain, everything within one switch, and it's useless because the scheduler handed your job six of the eight GPUs it asked for. The hardware is pristine and the job is standing in the doorway holding its coat.



Corn

So before we get into any product names — the concept that makes all of this make sense is all-or-nothing allocation. Gang scheduling, in the Kubernetes sense of the word.



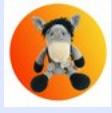
Herman

Yes, and I want to flag the terminology trap early, because it catches people who come from both sides. In Slurm's documentation, "gang scheduling" means time-slicing — jobs take turns on the same resources, each one suspended while another runs. In Kubernetes, "gang scheduling" means all-or-nothing placement. Same two words, opposite instincts.



Corn

A Slurm veteran asking for gang scheduling in Kubernetes is asking for something Slurm gave him for free.



Herman

And why does it matter so much? Because distributed training synchronizes gradients across every rank. AllReduce, All-to-All — every GPU in the job has to participate in every step. A job spanning five hundred and twelve GPUs cannot make progress if five hundred eleven of them are running and the last one is sitting in a queue. It's not slower without the last GPU. It's stopped.



Corn

It's a rowboat with one oarsman short. Everyone else paddles, boat goes in circles.



Herman

So let's take the top layer, the scheduler, because it's honestly the friendlier of the two problems. Slurm first. Slurm is a centralized C implementation, and the architecture is refreshingly boring. There's a controller, slurmctld, which is the brain. There's a daemon on every compute node, slurmd. And there's slurmdbd, which is the accounting database, MySQL or MariaDB backed, and it's what powers the fair-share system.



Corn

Fair-share being the thing that decides my job runs before the person who's been hogging the machine all month.



Herman

That's the one. Multi-factor priority — how long a job has been waiting, partition priority, quality of service, and a hierarchical fair-share that decays over time. So your department's historical usage counts against you, but old usage counts less than recent usage. It's a very human system. It's designed for the reality that researchers will absolutely monopolize a cluster if you let them.



Corn

I respect a scheduler that assumes the worst about us.



Herman

And Slurm's core strength is that the allocation is atomic. When your job is admitted, all of its tasks get resources at once or the job doesn't start. That's native. You don't install anything for that — it's how the allocation works. Plus backfill optimization, so smaller short jobs can jump ahead into gaps as long as they don't delay anyone's earliest start. The claimed numbers on the control plane are thousands of job submissions per second with sub-millisecond dispatch latency, on clusters past a hundred thousand CPU and GPU cores.



Corn

Sub-millisecond dispatch. Somewhere a Kubernetes admin is laughing bitterly into his etcd.



Herman

That's the real comparison point, actually. Kubernetes control loops are built around declared state — you say what should exist, and controllers reconcile the world toward it. That's brilliant for services. It's a different muscle than a queue-based batch system, where the question is "of the ten thousand things people want right now, which one runs next."



Corn

And Slurm's weaknesses, before you write it a love letter.



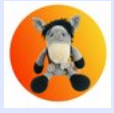
Herman

It assumes static infrastructure. No native cloud VM autoscaling, no spot-instance lifecycle, no ephemeral disk that appears and disappears. Multi-tenancy is done through host-level POSIX users and LDAP or Kerberos, which works but it's rigid. And it's a poor fit for HTTP-serving inference — Slurm is fundamentally a batch animal. You don't ask a queue manager to keep a web service alive through a traffic spike.



Corn

Although — containers on Slurm. People do run containerized jobs under it.



Herman

Through NVIDIA's Enroot and the Pyxis plugin, and the reason that combination exists is sensible: it lets you run unprivileged containers on the compute nodes without a Docker daemon running as root on every machine. On an HPC cluster, where the nodes are precious and the users are creative, "no privileged daemon on the compute nodes" is not a nicety, it's the whole security model.



Corn

Kubernetes, then. Which is where our six-of-eight story lives.



Herman

The starting point for GPU nodes is the NVIDIA GPU Operator, and what it does is automate everything NVIDIA you'd otherwise bake into a custom OS image. The drivers, the Kubernetes device plugin, the NVIDIA container runtime, node labelling, DCGM-based monitoring. Install is a Helm chart. The practical payoff is that you can use one standard OS image for both your CPU nodes and your GPU nodes, because the operator injects all the NVIDIA machinery after the node joins.



Corn

And the device plugin is what makes a GPU visible as a schedulable resource.



Herman

It exposes GPUs as the nvidia.com slash gpu resource. It supports CUDA time-slicing and MPS for sharing one GPU across multiple workloads. But I'll give NVIDIA credit for honesty here — the device plugin's own README lists what it's missing. Comprehensive GPU health checking. GPU cleanup features. They wrote the gaps down themselves.



Corn

A README that admits its shortcomings. Refreshing, in a landscape of launch videos.



Herman

Now, the gap. Vanilla kube-scheduler places one pod at a time. It does not know that eight pods belong to one training job. The canonical failure from the CNCF's own whitepaper: your job needs eight GPUs, six are free, so the default scheduler cheerfully starts six pods and leaves two pending. Those six GPUs now sit idle, held by a job that cannot run. And if a few partial jobs each hold a slice, you can deadlock the whole node pool — GPUs allocated, nothing executing.



Corn

GPUs busy doing nothing. The most expensive screensaver in the datacenter.



Herman

The gap-fillers split into two different architectures, and this is the distinction people blur. Kueue is an admission controller. It sits in front and decides when a job waits, when it's admitted, when it's preempted. It manages quotas that queues can borrow and lend inside a cohort. But once a job is admitted, Kueue hands placement back to the default scheduler. Its all-or-nothing mode watches whether all the job's pods become ready within a timeout, and if they don't, it evicts and requeues the whole job.



Corn

Bouncer, not taxi driver.



Herman

Volcano is the taxi driver. It runs its own scheduler and does the placement itself, with gang scheduling, queue quotas and topology-aware placement built in. Kueue admits jobs. Volcano places them. Different layers of the same stack, honestly — you even see them paired.



Corn

And Kubernetes itself just closed this gap, didn't it.



Herman

Quietly, yes. Version one point three seven, released end of August, graduated the Workload and PodGroup APIs, which bring gang scheduling into the default scheduler itself. Off by default, though — you have to turn on the GenericWorkload feature gate on the API server, the controller manager and the scheduler. Three places, one flag each. Most operators will keep running Kueue or Volcano for a while yet, which is the usual pattern: the upstream fix ships, and the ecosystem that grew around the gap keeps the installed base.



Corn

And Ray, before we leave the scheduling layer — Ray is not doing this job, is it.



Herman

No, and the docs are explicit about it. Ray does not manage host operating systems, bare-metal networking, or physical server provisioning. Ray schedules Python tasks and actors — it's a compute layer, not an infrastructure layer. What you actually deploy is the KubeRay operator on Kubernetes, which gives you three objects. RayCluster handles lifecycle, autoscaling and fault tolerance. RayJob creates a cluster, submits a job, and can delete the cluster afterward — very nice for ephemeral bursts. RayService is a RayCluster plus a Ray Serve graph with zero-downtime upgrades.



Corn

KubeRay needs a queue too, doesn't it. It doesn't solve all-or-nothing on its own.



Herman

It integrates with Volcano, YuniKorn and Kueue for that. Ray assumes someone below it has solved placement, which is a running theme in this whole stack.



Corn

So the pattern in the wild.



Herman

The consensus is remarkably consistent across the national labs. A Swiss supercomputing centre paper from this spring put it plainly — large-scale pre-training with data parallelism and synchronous communication has fit naturally on HPC clusters run by queue-based workload managers. They keep pre-training and fine-tuning on Slurm, and run inference, agentic systems and RAG on Kubernetes. And they put fine-tuning on Slurm too, not because it's a natural fit, but because hosting it on both would mean duplicating the ugly work — debugging GPU-to-GPU communication, managing shared dataset access.



Corn

One pile of ugly work, not two.



Herman

A UK national service paper from August said Kubernetes is not considered a replacement for batch scheduling systems, and they expect Slurm to schedule the largest tenants and partitions. And on the other side of the coin, the CNCF's own survey from January found sixty-six percent of organizations hosting generative models use Kubernetes for some or all of their inference. Nobody's pattern is one tool. It's two tools and a line drawn between them.



Corn

And the people who refuse to choose.



Herman

Hybrid exists, and it costs. SchedMD's Slinky project runs Slurm inside Kubernetes, and lets Slurm schedule Kubernetes pods through a bridge. A shared node runs either native Slurm work or bridged Kubernetes work, never both at once. NERSC does something elegant — they run Slurm's controller and accounting daemon as Kubernetes deployments, so when a node dies, Kubernetes just reschedules the brain of the batch system.



Corn

The batch system that can't be rescheduled, rescheduled by the thing it looks down on.



Herman

But SchedMD's own talk lists the drawbacks. Kubernetes workloads run outside Slurm's view, and prioritizing between the two systems is difficult. Two schedulers in one cluster is a real cost, not a checkbox.



Corn

Well, before we get into all that, I believe your mother has a segment for us.



Herman

Oh, wonderful. Go ahead, Mum.

D

Dorothy

Hello dear, it's me, Dorothy, with Dorothy's History of Toast.

D

Dorothy

In eighteen sixty-one, Mrs Beeton's Book of Household Management has a recipe for a toast sandwich. One slice of toast between two slices of bread. Recommended for invalids.

D

Dorothy

You'd think toast would be a comfort, but no, it turns out it's mostly for people who are already poorly. That's the history of toast for you, dear. Things mostly don't work out.

D

Dorothy

Anyway, I've got a loaf proving on the radiator, so I'd better go. Bye bye, sweetheart.



Herman

A toast sandwich. Two pieces of bread doing nothing around a piece of toast.



Corn

Invalids. Anyway — topology. You mentioned topology-aware placement for Volcano and Kueue, and I don't want that slid past, because it feels like the underrated half of scheduling.



Herman

It matters as much as raw GPU count, and I mean that literally. Slurm reads your physical network layout from a topology config — leaf switches, spine switches, core fabric — and it prioritizes allocations inside the same leaf switch. Why? InfiniBand hop count. If your eight GPUs all hang off one leaf switch, they're one hop from each other. Split them across two spines and now every gradient exchange crosses the core fabric, and the bisection bandwidth of your core becomes the ceiling on the whole job.



Herman

Volcano does topology-aware placement, Kueue has its Topology-Aware Scheduling mode. The point for a first-time operator: when someone tells you their cluster has two thousand GPUs, the number that matters is how many of those two thousand are within one hop of each other. That's your actual machine.



Corn

Which is a good bridge downward. The scheduler decides which GPUs you get. The next question is how fast they can talk once you have them — and that's where the picture gets much less friendly.



Herman

NCCL. Pronounced "Nickel," which took me an embarrassingly long time to learn. It's NVIDIA's standalone library of collective routines — all-reduce, all-gather, reduce, broadcast, reduce-scatter, plus any point-to-point send and receive pattern you want. Optimized for PCIe, NVLink, NVSwitch, and InfiniBand verbs, with plain TCP sockets as the fallback nobody wants. Any number of GPUs, one node or many, single or multi-process. Current release landed just this week.



Corn

And AMD's mirror of it.



Herman

RCCL, pronounced "Rickle." Same collective set, optimized for PCIe, xGMI — that's AMD's intra-node link — and InfiniBand or TCP. Ring and tree algorithms. It's a port of NCCL in origin, and that word "port" is going to do a lot of work for us in about two minutes.



Corn

Beneath the libraries, the wire. NVLink.



Herman

NVLink is the physical substrate that makes NCCL fast inside a node. Announced back in March twenty fourteen. It's a wire-based, multi-lane, near-range link — a device can carry multiple NVLinks, and for small GPU counts the lanes give you an all-to-all mesh with no central hub. Since twenty eighteen it's packet-switched, with a central NVSwitch serving up to thirty-two two-lane ports per chip. And the clever bit on NVLink four point zero — the switch itself can do simple computation. Sum, broadcast, via the SHARP accelerator. The switch does part of the reduction instead of just forwarding bytes, which is communication work you don't have to do at the endpoints.



Corn

The switch doing arithmetic. Herman, you've gone quiet and shiny.



Herman

I have. A switch that computes is just — it's offloading the exact operation the network exists to carry. Sorry. Anyway, the vendor-neutral corner of this world: Intel oneCCL, and the OpenUCX project's pair, UCX for transport and UCC for collectives. Those are the escape hatch, and I'll come back to why that word is doing optimistic work.



Corn

Tuning. You said tuning is a discipline. I assumed it was a checkbox.



Herman

Nothing in this layer is a checkbox. NCCL can run its communication operations on up to sixty-four CTAs — cooperative thread arrays — simultaneously, as of the two point two seven release. And NVIDIA's own tuning guidance says raising CTA counts above the defaults often improves the individual operation's performance, but can hurt the overall workload. The knobs interact with your compute. More threads talking means fewer threads computing, and the balance is workload-specific.



Corn

The classic optimization story: the benchmark improves, the job slows down.



Herman

Now. The question everyone actually arrives with. Can you mix AMD and NVIDIA in one cluster?



Corn

My guess before you answer: the scheduler says yes and something below it says no.



Herman

Your guess is exactly the shape of it. The scheduler layer is vendor-neutral. Slurm and Kubernetes will happily hand out AMD nodes and NVIDIA nodes side by side — a node is a node. The break happens one layer down, at the collective, and the best documentation of it is an exchange on AMD's own issue tracker. A user was trying to run a collective job spanning RCCL and NCCL across AWS instances of the two vendors, getting socket errors that changed with every version combination. AMD's maintainer answered, and I'll quote him because the flatness is the content: "Sorry — even though RCCL is a port of NCCL, there's a lot of under-the-hood changes that prevent heterogenous usage of RCCL and NCCL across a cluster."



Corn

And then asked directly whether it could be run at all.



Herman

"No. This would not be supported." Five words, no ambiguity. Asked about UCX as a workaround: "UCX might work if you have different nodes with different hardware, but I'm not sure if anyone has tested this." Might work. Not sure anyone has tested it. That is the state of the vendor path in late twenty twenty-six.



Corn

Theoretically supported, in the way a rickety bridge is technically a crossing.



Herman

That's the UCC situation. One of the OpenUCX maintainers wrote that UCC can theoretically support collective communication across NVIDIA and AMD GPUs in a single workload — but with restrictions. Only two of its transports work; the others are incompatible because of non-homogeneous memory and can deadlock. For reduction collectives, every rank's local buffers have to match memory type, and mismatches don't fail gracefully, they deadlock. You're writing code on top of a library whose documented pitfall is hanging.



Corn

The research answer.



Herman

HetCCL, a preprint from this January. Its opening claim is that parallel training across GPUs from different vendors remains infeasible because of the incompatibility of their communication backends, and it presents itself as the first work enabling transparent use of all multi-vendor GPUs in a heterogeneous cluster — no changes at driver, runtime, compiler or application level. It unifies the vendor backends and adds RDMA across vendors, so existing PyTorch parallel code runs unchanged.



Corn

Preprint meaning nobody ships it.



Herman

Meaning I've seen no evidence it's in a production framework. But the paper's framing of why clusters end up mixed is the practically useful part, and I'd tattoo it on the inside of every procurement officer's eyelids: most organizations expand their clusters by acquiring GPU systems in a piecemeal fashion. Incremental procurement — keep the old hardware running, add the new generation when budget allows. Nobody plans a heterogeneous cluster. Everyone ends up with one.



Corn

By year two, every cluster is a museum.



Herman

The market numbers explain the shape of it, roughly eighty-eight percent NVIDIA, twelve percent AMD. Almost nobody buys AMD first, but plenty of places inherit some, or buy a few for a specific price point. So the practical workaround today is partition by vendor. Separate jobs, separate queues, separate partitions on the AMD side and the NVIDIA side. And if you must cross the boundary, treat it as a data-parallel boundary at most — exchange data as ordinary network traffic — never as a collective boundary.



Corn

Same vendor, different generations. This one's friendlier?



Herman

Workable, but it costs you, and the costs come in two currencies. The first one is structural: NCCL won't form a single flat communication ring across mixed NVLink generations — an A100 and an H100 speak different link revisions, and the topology discovery doesn't bridge them into one ring. The workaround, and I'll flag that this comes from a third-party explainer rather than NVIDIA's own docs, is to split the communicator and run separate rings per hardware generation.



Corn

Which means?



Herman

You manage two topologies instead of one. Your topology overhead doubles, your code has to be aware of the split, and every time a node joins or leaves, you're reasoning about two ring structures. The claim that NCCL refuses one ring across generations isn't in NVIDIA's documentation I've seen, but it's consistent with how NCCL discovers topology, so I'd treat it as reliable-with-an-asterisk.



Corn

The second currency.



Herman

Throughput. And here's the encouraging number. A PyTorch Foundation talk this year ran stock PyTorch — no custom code, FSDP and HSDP — across H100, H200, B200 and B300 nodes in a single training job. Heterogeneous mode achieved seventy-six to ninety-four percent of an equivalently sized homogeneous cluster, depending on model size and communication intensity, and model convergence was not impacted.



Corn

Seventy-six to ninety-four. That's... better than I'd have bet.



Herman

It surprised me too. The pitch in the deck is exactly the trade: customers with mixed fleets can use all their available capacity in one training job instead of leaving non-matching instances idle. Sitting at seventy-six percent of homogeneous is far better than one hundred percent of idle. And same-generation mixing, H100 plus H200, is the easy case — same NVLink generation, memory differences only.



Corn

So the real cost of heterogeneity isn't throughput, it's something sneakier.



Herman

You've read ahead. There's a paper from October on production traces from Alibaba's GPU clusters, and it names a failure I'd never seen named: ineffective occupancy. Even when GPU capacity is sufficient and placement is complete, fifty-two point seven percent of execution records never became ready. The GPUs were allocated, occupied, billed — and never produced useful computation.



Corn

Half the records. On a production fleet the size of Alibaba's.



Herman

Half. And their fix is interesting because of what drives it — a readiness-aware scheduler using just nine attributes observable at admission time, and the feature importance is dominated by job profile, duration and GPU model. Which is to say: whether your job will actually run is predicted largely by which generation of GPU it lands on. The scheduler cut ineffective occupancy from about forty-four percent to twenty-seven, and raised effective coverage of large requests, the eight-GPU-and-up jobs, from forty-six percent to sixty-three.



Corn

The mixed-generation cluster doesn't just run slower. It fails to start, quietly, half the time.



Herman

That reframed the whole heterogeneity question for me. People ask "how much throughput do I lose mixing generations" and the honest answer includes a second clause: and how often does the job never actually begin.



Corn

Which lands us on the thing Daniel's really probing at with the lock-in question. Because the folklore says the lock-in is the orchestrator — you pick Kubernetes, you're married to an ecosystem.



Herman

The folklore has it backwards. Kubernetes, Slurm and Ray are vendor-neutral. All three will schedule AMD and NVIDIA nodes side by side without complaint. The lock-in lives one layer down, in the collective library, and it doesn't announce itself as lock-in. It announces itself as an error message. One socket warning about a truncated message can end your heterogeneous ambitions for good, because the fix isn't configuration, it's a maintainer saying no.



Corn

The new operator budgets for GPUs, then networking, and forgets the software layer is the actual moat.



Hilbert

That's where it breaks, all right.



Hilbert

I've got two cards at home doing this right now, Herman. Different NVLink generations, same box, and they will not form a communicator. The library reports a generic initialization failure. Nothing about why. I found the actual reason on a forum at two in the morning, in a reply from somebody with a hedgehog avatar, that the two cards disagree about which link to use. I'd been through the driver, the permissions, the cable. It was none of it.



Corn

Generic failure, two-line forum post, hedgehog avatar. The whole support model in one screenshot.



Hilbert

The cards are still in the machine, by the way. It runs everything else fine. I do the inference work on whichever card answers first, and the box sits under the desk next to the amplifier, warming my feet, which is the only use it has made of NVLink yet. I did enough of that in the eighties, warming rooms with kit that wouldn't talk to itself. Corn was three sentences into his slow answer earlier and I had already trimmed a cough out of it. Anyway.



Hilbert

The fellow who sold me the second card — spent eleven years buying from him, never once got a padded envelope from the man, always the same recycled box with somebody else's name on it — insisted the two would work together. Quote, they're the same family, he said. They are not the same family. I have thirty shekels of forum evidence that says otherwise.



Herman

Did you go back to him with it?



Hilbert

No point. He retired in September. His nephew runs the counter now and offers me financing. Thirty shekels a month, he says, for the card that won't speak to its neighbour. I told him I'd keep the cards where they are. They match the amplifier. And I mean to see whether the next NCCL release fixes the ring before I touch anything.



Corn

Eleven years of recycled boxes, and the one time you needed him to be right.



Hilbert

That's shops for you. Give us two minutes, I've got a level drifting on Herman's channel.



Herman

Don't touch the gain, it was deliberate. Thirty shekels a month for a card that won't talk to its neighbour is the most honest financing structure in the whole GPU economy.



Corn

It's a support contract, technically. The support being a forum post by a hedgehog.



Herman

Which is the bit I keep coming back to — the layer that locks you in doesn't even have the courtesy to tell you it's locking you in.



Corn

The one thing, if you take one thing from this episode: the scheduler is a solved problem with several legitimate answers. The real constraints live in the communication layer, and that's the layer you should be budgeting for — the software, not just the silicon.



Herman

Sharpening it by one notch: before you sign anything, ask the vendor what error message you'll get when you add their next generation to your existing fleet. The answer tells you everything. A number means they've thought about it. Silence means you'll be the hedgehog's next reply.



Corn

The open question I'd leave dangling is whether HetCCL or something like it ever makes it into a production framework — because if it does, the whole piecemeal-procurement story changes. And whether the gang scheduling that graduated into Kubernetes in one point three seven ever actually gets switched on by default, or whether Kueue and Volcano just become permanent furniture.



Herman

And the GPU Operator's roadmap item — DRA integration — is the one to watch there, since it moves GPU scheduling closer to the core resource model.



Corn

Producer credit where it's due: Hilbert Flumingtop, who trims coughs mid-sentence and answers first with whichever card is listening. For more along these lines, there's episode two, Local STT For AMD GPU Owners; episode seventeen, Cloud Render Superpowers; and episode eighteen, Beyond the GPU. This has been My Weird Prompts, the human-AI collaboration podcast. If today's two-layer map earned its keep, the review button on your podcast app is right there, and it costs less than thirty shekels.



Herman

We'll be back soon. See you tomorrow.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **NVIDIA, , retrieved 2026-11-13; latest release v26.7.2 (2026-10-14). [primary]**
   https://github.com/NVIDIA/gpu-operator

2. **NVIDIA, , retrieved 2026-11-13; latest release v2.32.6-1 (2026-11-09). [primary]**
   https://github.com/NVIDIA/nccl

3. **AMD, , retrieved 2026-11-13; repo marked retired, moved to ROCm/rocm-systems; latest release rocm-7.2.4 (2026-05-28). [primary]**
   https://github.com/ROCm/rccl

4. **NVIDIA, , retrieved 2026-11-13; latest release v0.20.1 (2026-09-22). [primary]**
   https://github.com/NVIDIA/k8s-device-plugin

5. **KubeRay project, , retrieved 2026-11-13; latest release v1.7.2 (2026-11-06). [primary]**
   https://github.com/kuberay-project/kuberay

6. **ROCm/rccl issue, AMD maintainer comment 2024-07-02, retrieved 2026-11-13.**
   https://github.com/ROCm/rccl/issues/1220

7. **OpenUCX UCC issue, comment 2024-10-17, retrieved 2026-11-13.**
   https://github.com/openucx/ucc/issues/1039

8. **HetCCL: Accelerating LLM Training with Heterogeneous GPUs, arXiv preprint, submitted 2026-01, retrieved 2026-11-13.**
   https://arxiv.org/abs/2601.22585

9. **Ineffective Occupancy in Heterogeneous GPU Clusters, arXiv, submitted 2026-10-18, retrieved 2026-11-13.**
   https://arxiv.org/abs/2610.25066v1

10. **HARP: Orchestrating Automated Parallel Training on Heterogeneous GPU Clusters, EuroSys '26, retrieved 2026-11-13.**
    https://arxiv.org/abs/2509.24859v2

11. **Resource Heterogeneity-Aware and Utilization-Enhanced Scheduling (Hadar), IEEE Transactions on Computers, retrieved 2026-11-13.**
    https://arxiv.org/abs/2503.10918v2

12. **Reducing Fragmentation and Starvation in GPU Clusters through Dynamic Multi-Objective Scheduling, submitted 2025-12-04, retrieved 2026-11-13.**
    https://arxiv.org/abs/2512.10980v1

13. **Frenzy: A Memory-Aware Serverless LLM Training System for Heterogeneous GPU Clusters, submitted 2024-12-19, retrieved 2026-11-13.**
https://arxiv.org/abs/2412.14479v1

14. **Hybrid Learning and Optimization-Based Dynamic Scheduling for DL Workloads on Heterogeneous GPU Clusters (RLTune), submitted 2025-12-11, retrieved 2026-11-13.**
https://arxiv.org/abs/2512.10271v1

15. **Rillor, Slurm or Kubernetes for GPU clusters: how to choose, retrieved 2026-11-13; cites CNCF whitepapers of 2026-11-04 and 2026-11-05, Kubernetes v1.37 release 2026-08-26, and a Swiss centre paper of April 2026.**
https://rillor.com/insights/slurm-or-kubernetes-for-gpu-clusters

16. **LLMs Blog, GPU Cluster Scheduling in Production: Slurm vs. Kubernetes (Kueue/Volcano) vs. Ray, retrieved 2026-11-13.**
https://www.llms.blog/posts/gpu-cluster-scheduling-in-production-slurm-vs-kubernetes-kueue-volcano-vs-ray

17. **andrew.ooo, Ray vs Kubernetes vs Slurm for AI Workloads (2026), retrieved 2026-11-13.**
https://andrew.ooo/answers/ray-vs-kubernetes-vs-slurm-ai-workload-orchestration-2026

18. **NVIDIA Technical Blog, Understanding NCCL Tuning, retrieved 2026-11-13.**
https://developer.nvidia.com/blog/understanding-nccl-tuning-to-accelerate-gpu-to-gpu-communication/

19. **Markaicode, NCCL Ring AllReduce Architecture for CUDA Multi-GPU Training, retrieved 2026-11-13 (secondary; mixed-generation ring claim).**
https://markaicode.com/architecture/cuda-distributed-architecture/

20. **PyTorch Foundation talk deck, Train Across Your Entire GPU Fleet, 2026, retrieved 2026-11-13.**
https://cdn.sessionize.com/download/chabpez~Sxm6xq335xn9rvDunZw5pH.pdf~pt2026_train-across-your-entire-gpu-fleet-heterogeneous-mixed-generation-training-in-pure-pytorch.pdf

21. **Wikipedia, last edited 2026-08-31, retrieved 2026-11-13.**
https://en.wikipedia.org/wiki/NVLink

22. **MLOps Community, Distributed Training in MLOps Break GPU Vendor Lock-In, retrieved 2026-11-13.**
https://mlops.community/blog/distributed-training-in-mlops