KubeCon CloudNativeCon North America 2026

#KubeCon #CloudNativeCon

Many Clusters, One GPU Pool

Locality-aware GPU scheduling with MultiKueue and Dragonfly

Yiqing Wang · Senior Software Engineer, Roblox

Adam Buran · Principal Software Engineer, Roblox



BEFORE

Where we started





us-east-1

one scheduler

25%

of allocated GPU time busy

6.05

preemptions per 100 GPU-days



Limited capacity
only us-east-1 GPUs

Workload portability
jobs stuck in one cluster

Data locality
placement ignored data

THE PROBLEM

Scattered GPUs

us-east-1

mostly B200

us-west-2

H200 + B200

us-east-2

B200 + RTX PRO 6000

on-prem

H200 + RTX PRO 6000



Data lives in one region.

Uneven load

Idle and queued at once.

Quota stuck per cluster

Quota couldn’t follow jobs.

Placement by hand

Users picked clusters.

DESIGN CHOICE

Solution Space

	YuniKorn per cluster SINGLE-CLUSTER	Karmada FEDERATION	Kueue + MultiKueue MULTI-CLUSTER
Tiering	✓	build	✓
Preemption	✓	build	✓
Sharing idle GPUs	✓	build	✓
Locality-aware dispatch	✗	partial	plug-in
Gang-scheduling	✓	alpha	✓
	Single cluster only	We'd build 3 controllers	✓ Chosen: upstream core

Design Principles

1. **Stay upstream**
   Stock releases plus our glue.



2. **Isolation**
   A failure stays in one spoke.



3. **Soft locality**
   Data region first. Anywhere after 5 minutes.

4. **Fail open**
   Failures slow jobs, never stop them.



5. **Resource accounting**
   Measure per allocated GPU-hour.



WHAT IT BOUGHT US

Results

+80% useful compute per allocated GPU-hour vs. legacy cluster

WHERE JOBS RUN

89%

of jobs ran near their data

WHO GETS CAPACITY

−80%

preemptions per GPU-day

WHEN A SPOKE FAILS

9 in 10

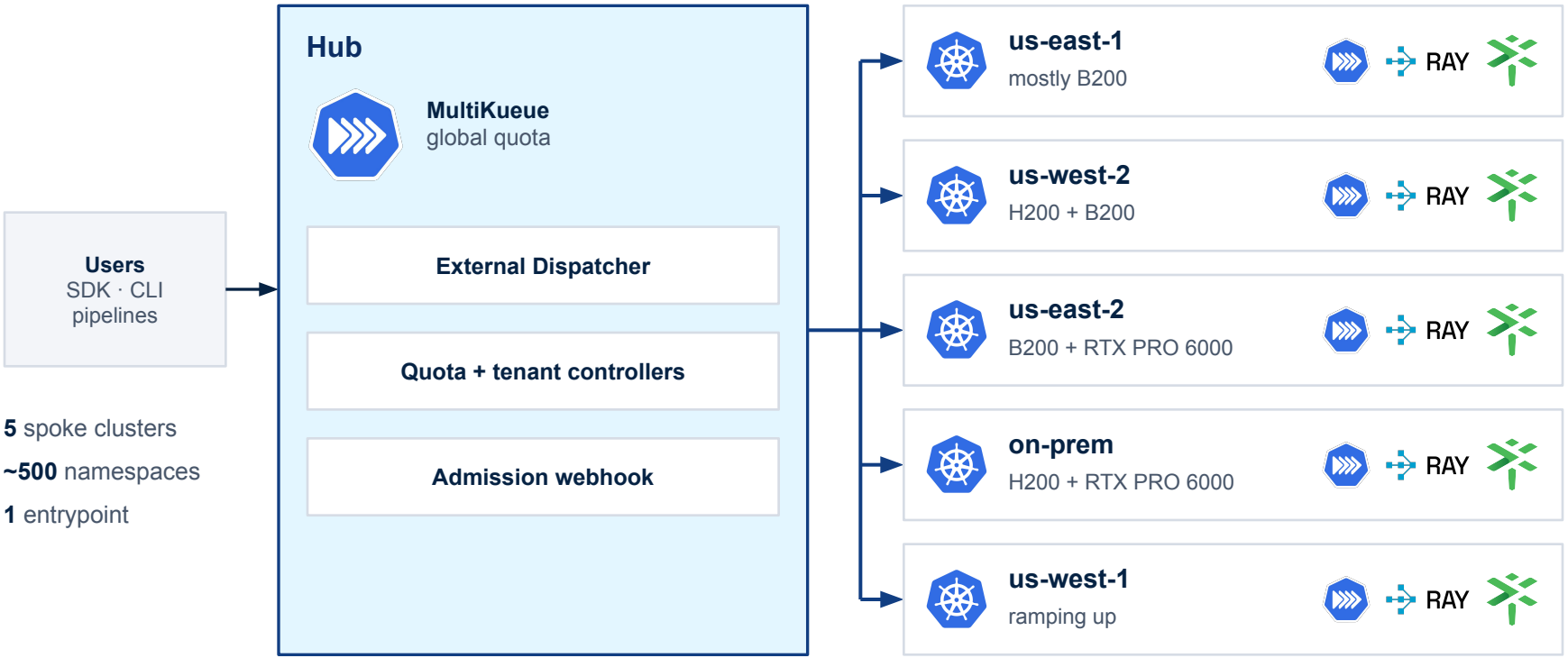
new jobs ran elsewhere

HOW DATA ARRIVES

91%

of bytes skipped S3

Hub and Spoke



How jobs get placed

```yaml
apiVersion: ray.io/v1
kind: RayJob
metadata:
  name: train-shape-vae
  labels:
    mlp.roblox.com/instance-type: p6-b200.48xlarge
    kueue.x-k8s.io/priority-class: tier0
    mlp.roblox.com/data-locality: us-east-1
```

No cluster name needed

Hub
knows quota

Spokes
know their nodes

1. **Hub reserves quota**
   Quota across all spokes
2. **Dispatcher nominates spokes**
   By capacity and data region
3. **A spoke admits it**
   Checks real nodes. First wins.
4. **KubeRay runs it**
   Status syncs to the hub

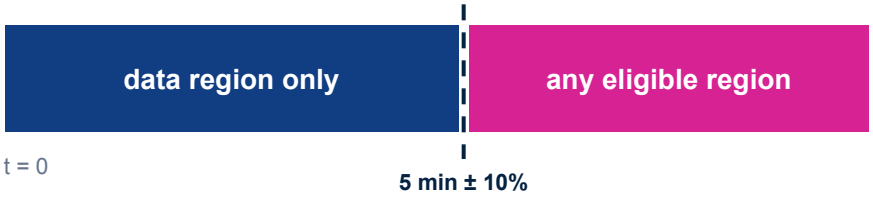
Eligibility before locality

EVERY SPOKE MUST PASS

- Allows the framework
- Has the GPU type
- Exposes EFA or RDMA if asked
- Can mount the volumes
- **Fits the whole gang**

Blocks ~5K partial gang fits a day.

THEN PREFER THE DATA'S REGION



89%

ran in their data region

2.4 s

median submit-to-admission

Long waits cost more than remote reads.

WHO GETS CAPACITY

Three capacity tiers

Guaranteed

Team-owned quota. Idle GPUs get lent.

Shared

Runs on lent GPUs. Preempted first.

Opportunistic

Spot instances. No quota.

ONE TEAM'S GUARANTEED QUOTA

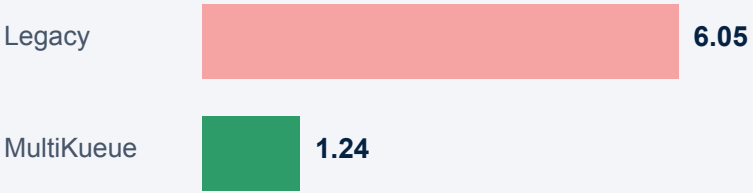


Defined once. Rendered everywhere.

PREEMPTION RESULTS

80% fewer preemptions

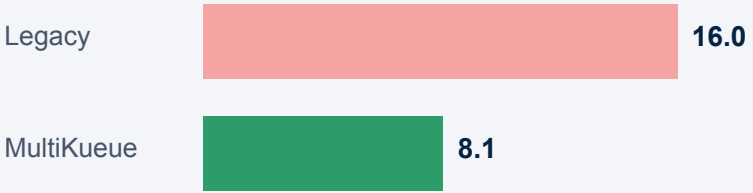
Per 100 allocated GPU-days



−80%

All tiers with 45% more work.

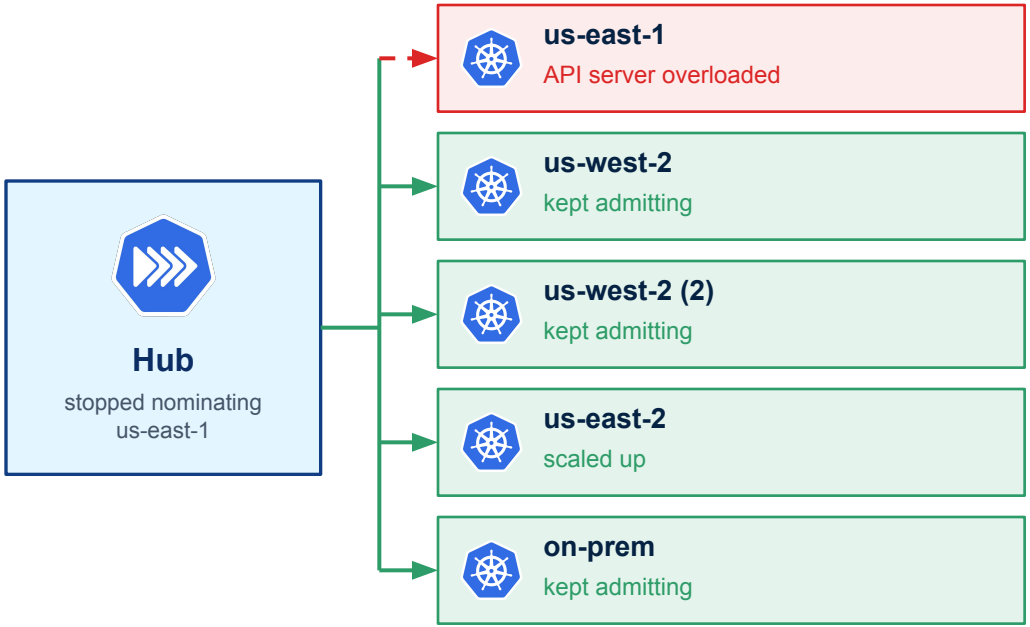
Per 100 preemptible GPU-days



−49%

Preemptible work only.

Surviving a spoke outage



9 in 10

new jobs ran elsewhere

~1 min

to reroute requeued jobs

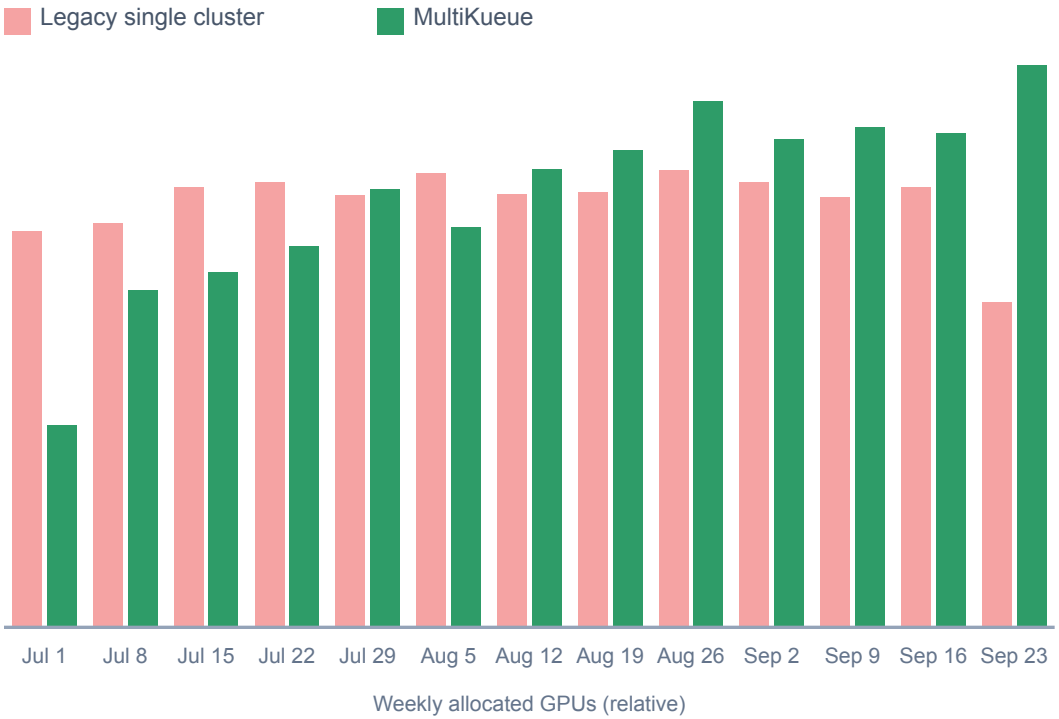
2×

GPUs in use on us-east-2

The outage stayed in one spoke.

ADOPTION

Most load moved over



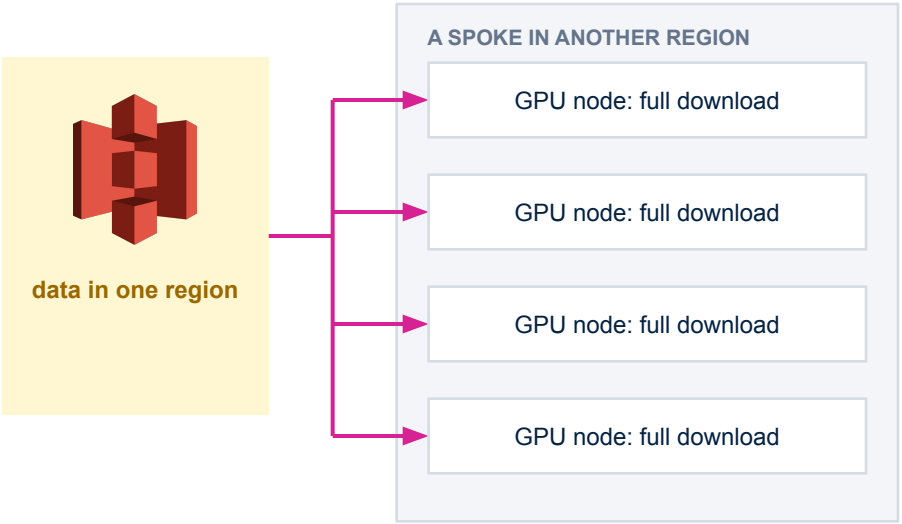
34% → 63%

MultiKueue share of load

53%

of 43K GPU workloads ran outside us-east-1

Data became the bottleneck




```
transfer = size × nodes × runs
```

~200 MiB/s

per node from S3

92.5%

of S3 bytes were repeats

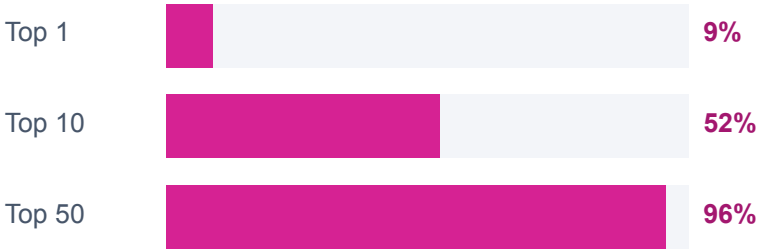
56%

were cross-region repeats

DATA DESIGN

Cache the hot set

A FEW DATASETS DRIVE MOST READS



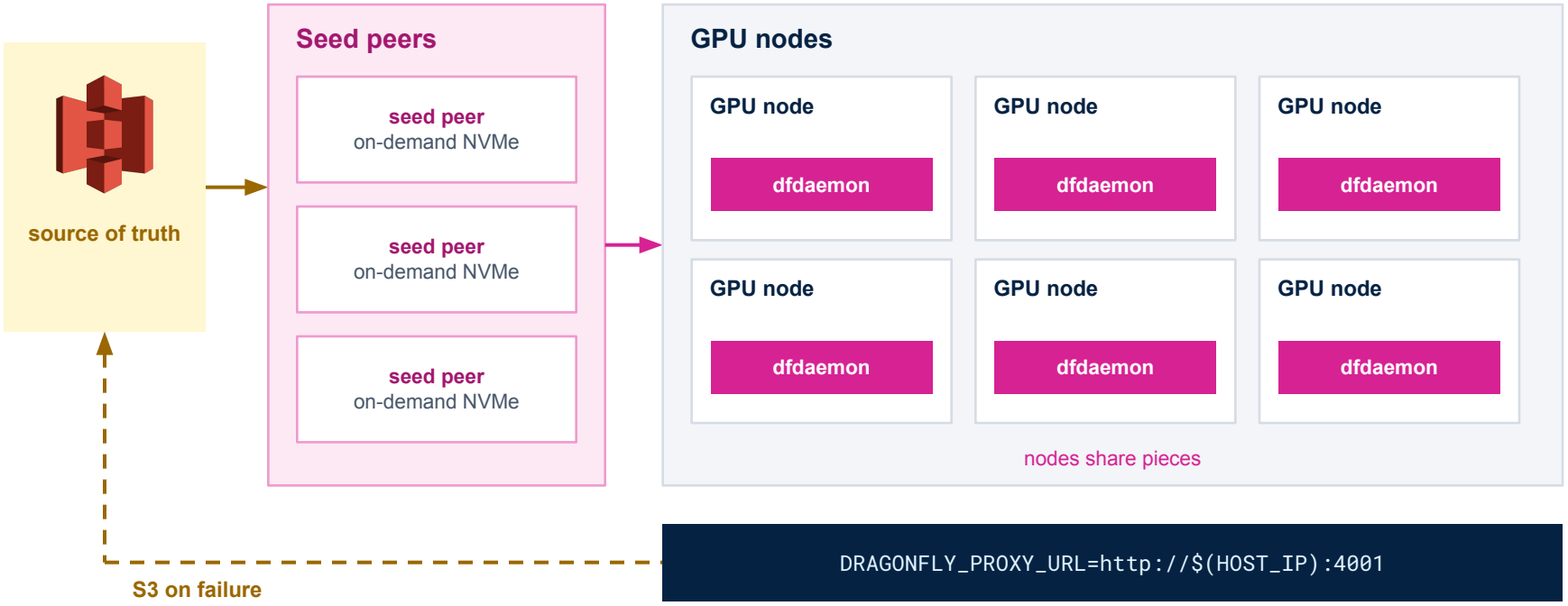
~15 TB of hot data drove **~3,500 TB** of repeat downloads.

WHAT WE WANTED

- ✓ **S3 stays the source of truth**
- ✓ **Opt in with one env var**
- ✓ **Any failure falls back to S3**
- ✓ **One cache per spoke**

Dragonfly met all four. Each byte crosses the WAN about once per spoke.

P2P with S3 fallback



DATA RESULTS

9 PB delivered

Training bytes by source since April



- **Same node's cache**
- **Peers in the spoke**
- **Straight from S3** 7%

Fastest 10 minutes in one spoke

51 GiB/s

us-east-1 with zero from S3

~1.5 GiB/s per node

91%

of bytes skipped S3

11×

bytes delivered per byte from S3

96%

from cache or peers in us-west-2

INFERENCE

8.8× faster model loads

Loading a 433 GiB model

Straight from S3



23.3 min

Via Dragonfly



2.6 min

8.8× faster

Second load hit the local cache.

~1 PB

to inference since May

91%

of bytes skipped S3

WHO LOADS MODELS THROUGH IT

Serving replicas

Each new replica loads the whole model.

Benchmark rounds

1,352 runs in 10 weeks.

Ferry staging

Weights arrive before the pod.

Data follows the job

Volumes move zones

Bytes are checked before the source is released.

Verified cross-AZ on kind

Data gates placement

A spoke is nominated once its data is staged.

Feature-gated

Whole jobs move

Checkpoint and data travel together.

Spot reclaim can trigger it

No guessed routes

Cross-region and EFS moves are refused.

Every refusal is in status

Preload Data

AHEAD OF THE POD

1. Weights

Dragonfly fills the node from peers.

170 GiB in 58 s

2. GPU checkpoint

Ferry copies and verifies it.

199 GB in 5 min

WHEN THE POD STARTS

3. Restore

The pod resumes from it.

healthy in 18 s

Time to first healthy check



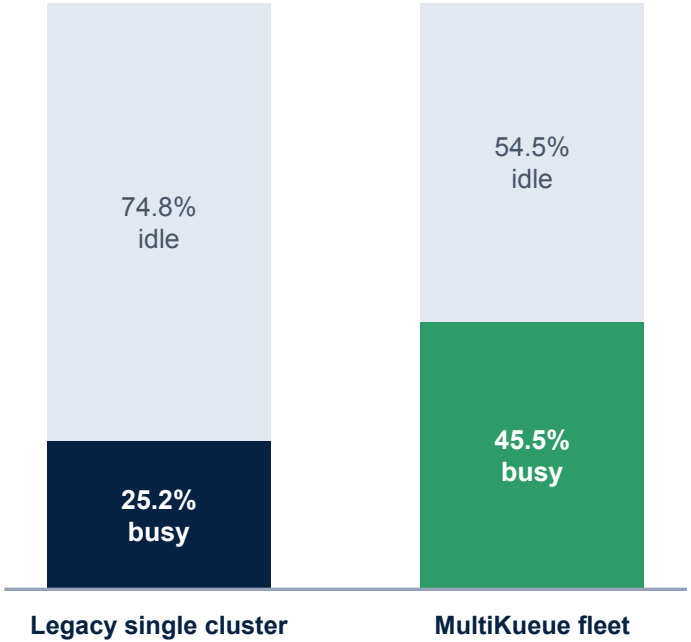
17×

faster start (pilot)

IMPACT

80% more useful compute

Busy share of allocated GPU time



+162%

productive GPU-hours

+6%

idle GPU-hours with 45% more work

Built from upstream parts



One entrypoint	Kueue everywhere. MultiKueue on a hub.
Who gets capacity	Tiers as priority classes in one cohort.
Where jobs run	Start built-in. Add locality and gang fit.



How data arrives	Dragonfly peers with S3 fallback.



Measure useful work	Busy GPU-hours and preemptions per GPU-day.

 **v0.19.5**

 **v2.5**

 RAY **v1.6**

Our glue: dispatcher, quota and tenant controllers, webhook

LESSONS

Learnings

01

Hub-only admission checks

On spokes they stall admission.

02

Validate on submit

Spokes reject names over 47 chars.

03

Labels don’t propagate

Allowlist them in labelKeysToCopy and kube-state-metrics.

04

Expire stale signals

Ignore reports older than 2 minutes.

05

Keep policy identical everywhere

A drifted Kyverno rule blocked RayJobs. Policy syncs from Git now.

06

Caches on stable nodes

Seed peers never run on spot.

CHALLENGES

Four open problems

Wide nominations waste copies

About 1 in 5 copies is thrown away.

Plan: best spoke first

Stranded gang pods

Scattered free GPUs strand a gang's last pods.

Plan: Topology-Aware Scheduling

RAY Preemption needs checkpoints

Reclaim can kill hours-long Shared jobs.

Plan: checkpoint on SIGTERM

Manual spoke onboarding

Secrets and config are hand-written.

Plan: turn on auto-registration

Stage data first



Pre-warm caches

CRDs deployed

Hot datasets stay in seed peers.



Stage data before dispatch

feature-gated

Dispatch once data is ready.

Cost-aware spot placement

observe-only

Rank spot by price per node-hour.



RDMA weight distribution

exploring

Dragonfly's RDMA path for serving and sandboxes.

TAKEAWAYS

What to take home

1

Pool behind one hub.

Outages stay in one spoke.

+80%

useful compute per GPU-hour

2

Give locality a deadline.

Data region first. Anywhere after 5 minutes.

89%

of jobs ran next to their data

3

Fetch once per spoke.

A P2P cache per spoke.

91%

of bytes skipped S3

KubeCon CloudNativeCon
North America 2026

Thank you!

Questions?

Yiqing Wang: https://www.linkedin.com/in/yiqingw/

Adam Buran: https://www.linkedin.com/in/adam-buran/

Many Clusters, One GPU Pool: Locality-Aware Scheduling and Data Delivery With Kueue and Dragonfly

