

AI, HPC, возможен ли универсальный кластер?

Роман Семенов
Системный архитектор

Проблема «Двух Миров»



Почему нельзя просто запустить AI на HPC (и наоборот)

Характеристика	HPC (симуляции)	AI (обучение LLM)
Точность	FP64 — критична (расходимость за миллион итераций)	BF16 / FP8 — низкая точность ради throughput (8–16× на Tensor Cores), шум используется как регуляризация
Bottleneck	Latency памяти, скорость CPU	Пропускная способность HBM и сети
Трафик	MPI, графовый обмен, мелкие пакеты	All-Reduce / All-to-All, массивные сообщения
Хранилище	Линейная последовательная запись файлов с результатами	Random read миллиардов мелких файлов
Сеть	1–2 IB-порта (200–400G)	4–8 IB (multi-rail, 3.2 Tbps)

Один GPU умеет обе ветви — у H100 есть и FP64-блоки, и Tensor Cores. В кремнии различий нет — они в сети и топологии узла. Универсальность — это не «одинаковое железо», а способность одного пула обслуживать оба профиля.

Спектр нагрузок



Между физикой и ChatGPT

Класс	Точность	Масштаб	Коммуникация	Режим
HPC (CFD, климат, молекулярка)	FP64	1–64+ узла	MPI, 1–2 IB	Постоянно
Inference быстрых моделей	FP16 / INT8	1 GPU / MIG	редко	24/7
Inference Frontier AI моделей	FP16 / INT8	8 GPU	nvlink, NCCL, 1 узел	24/7
Fine-tuning (LoRA)	BF16	1 узел, 8 GPU	nvlink, NCCL, 1 узел	Регулярно
LLM pre-training (frontier)	BF16 / FP8	64–1024+ GPU	All-Reduce, multi-rail	Редко, долго

HIGH PRECISION

HPC (CFD,
Молекулярка....)

ТОЧНОСТЬ

FP64 (Double)

МАСШТАБ

1-64+ Узлов

MPI, Постоянный режим

DEPLOYMENT

Inference Fast

МАСШТАБ

1 GPU (MIG) → 1 Узел

РЕЖИМ

24/7 Real-time

Light модели

DEPLOYMENT

Inference Frontier

МАСШТАБ

1-8 GPU → 1 Узел

РЕЖИМ

24/7 Real-time

Frontier модели

ADAPTATION

Fine-tuning (LoRA)

РЕСУРС

1 Узел

УСКОРИТЕЛИ

8 GPU

Регулярные запуски

MASSIVE SCALE

Pre-training

ТОЧНОСТЬ

МАСШТАБ

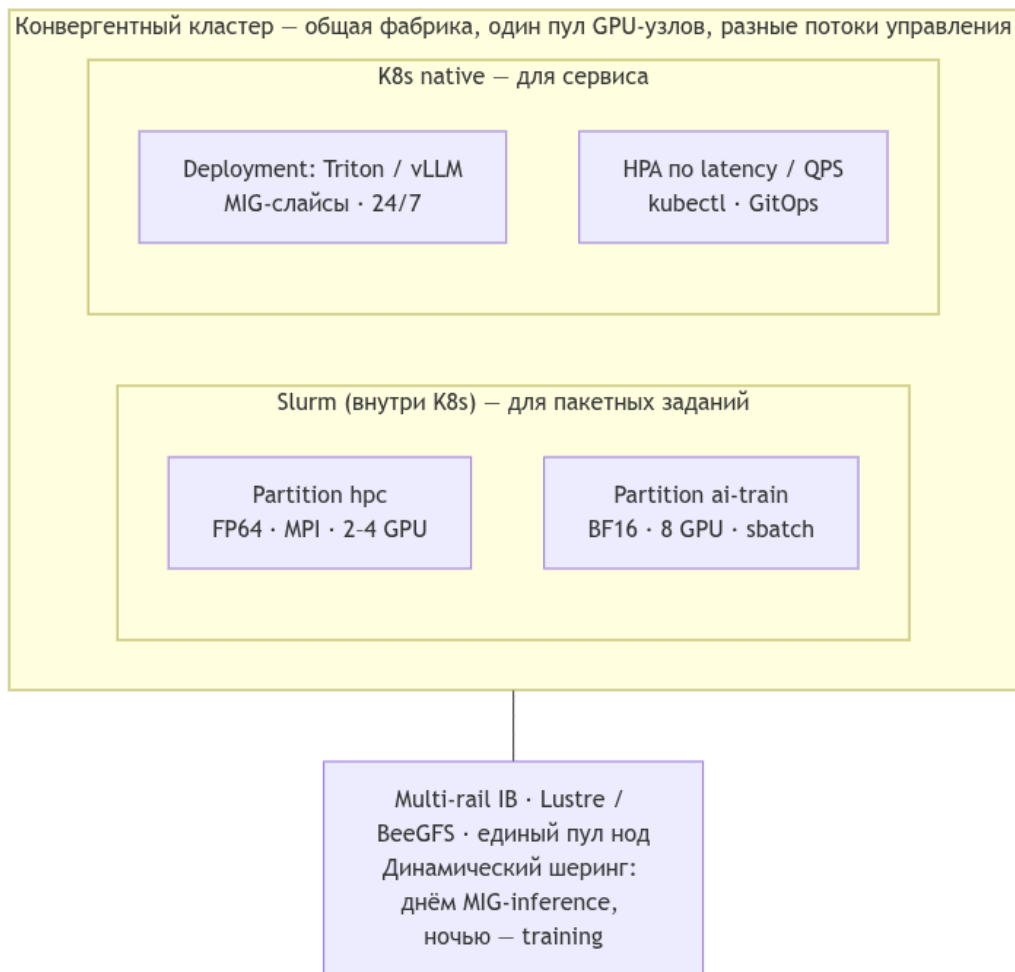
1024+ GPU

Длительные сессии

Универсальность = оркестрация



На одной фабрике — два пути управления



- Slurm в K8s — для батча: HPC-симуляции и обучение моделей. Partitions hpc, ai-train.
- K8s напрямую — для сервиса: инференс как Deployment, MIG-слайс, HPA.
- Координация на уровне узлов: Slurm знает, какие ноды заняты инференсом, и не трогает их. K8s не планирует поды на ноды, занятые sbatch-задачей.
- Профиль инференса задаёт размер ночного окна. Внутренний/корпоративный инференс - рабочие часы. Ночью - почти весь GPU уходит под Slurm fine-tuning и эксперименты.

Четыре причины разместить Slurm в Kubernetes

- Один ops-стек. Inference уже в K8s (Triton, vLLM, KServe, GitOps). Slurm рядом — одна команда админов, единый мониторинг. На bare-metal Slurm — два параллельных стека (Slurm-Ansible + K8s-Helm).
- НА из коробки. slurmctld в поде — K8s перезапускает при сбое.
- GPU-aware экосистема K8s. MIG, Topology Spread, DRA, GPU Operator, device-plugin — всё уже есть. Не нужно изобретать.
- Operator-паттерн. SlurmCluster (Controller) CRD — декларативный YAML. Изменения — коммит в Git. Откат — kubectl rollout undo. Bare metal — pdsh + ручные правки slurm.conf.

Инференс всё равно живёт в K8s. Slurm рядом на bare metal = два стека. Slurm в K8s = один стек.



Soperator / SUNK — паттерны эксплуатации



Первые проекты реализации идей slurm в k8s

Проект	Что взять	Ограничение
Soperator (Nebius)	jail — общий root FS через PVC (мгновенное обновление бинарников). ActiveCheck CR — пробы GPU/дисков/сети с автодрейном. Ephemeral nodes — переиспользование GPU между training/inference. SSSD (LDAP/AD), Tailscale, Prometheus/Grafana/Loki из коробки.	Single-partition — один SlurmCluster CR = одна partition (hpc ИЛИ ai-train); для multi-partition нужен форк оператора (roadmap без публичных сроков)
SUNK (CoreWeave)	Syncer — двунаправленный sync pod↔node (not-ready pod = drained Slurm-узел). Автоген topology.conf из K8s labels. GPU straggler detection — выявление отстающих GPU в multi-node training.	По умолчанию рассчитан на CKS (CoreWeave Kubernetes Stack, проприетарная K8s-платформа CoreWeave); для on-prem — режим SUNK Anywhere, IB-драйверы в образах надо верифицировать

Soperator и SUNK — production-форки от Nebius и CoreWeave. Доказывают, что «Slurm в K8s» — индустриальная практика, не эксперимент.

NEBIUS



Slinky – operator как upstream-основа



SchedMD (разработчик Slurm) + NVIDIA

Slinky – два независимых компонента

Компонент	Что делает	Когда выбирать
slurm-operator	Запускает slurmctld, slurmd, login, db, rest-api внутри K8s как pods. CRD nodesets/loginsets/accountings	У вас один K8s-кластер, нужны partitions hpc и ai-train под sbatch. Стандартный путь.
slurm-bridge	Slurm остаётся снаружи K8s как scheduler. K8s-Pod транслируется в placeholder-job Slurm. Slurm выделяет узлы и сообщает обратно.	У вас уже есть внешний Slurm, и вы хотите, чтобы K8s-нагрузки шли через него.

Версии: Slurm ≥ 25.11 , K8s ≥ 1.29 (operator) / ≥ 1.35 (bridge).

HA: рестарт pod'a (operator) – K8s перезапускает slurmctld, контроллер поднимает заново. Без shared FS.

Контейнеры: Job Containers – это OCI-образы, не нужен Puxis.

Установка: Helm chart, cert-manager, 4 команды.

Slinky – это upstream (не fork)



Slinky slurm-bridge – альтернативный путь



Slurm как scheduler для K8s, но Slurm снаружи

Этап	Что происходит
1	Пользователь отправляет обычный манифест K8s-Pod / JobSet
2	Slurm-bridge перехватывает, транслирует в placeholder-job Slurm
3	Slurm выделяет физические узлы с учётом очереди и fairshare
4	Bridge делает сажает Pod на выделенные ноды, kubelet запускает контейнеры

Когда использовать: у вас уже есть Slurm рядом с K8s, и вы хотите K8s-нагрузки отправлять через него, а не дублировать планировщик.

Ограничение: эксклюзивное выделение целого узла на Pod. Не подходит для плотных inference-сервисов с MIG-слайсами на одной ноде.

Совместимость: Slurm ≥ 25.11 , K8s ≥ 1.35 , DRA-only (CPU/GPU; IB через SR-IOV отдельно).



Volcano · Kueue · Ray — когда Slurm не нужен



K8s-native альтернативы самому Slurm

Проект	Тип	Сильная сторона	Ограничение
Volcano (CNCF incubating)	K8s-native batch	Gang/topology-aware scheduling, сильный для Spark / Ray / PyTorch	Нет MPI-семантики, fairshare по TRES, нет аналога slurmdbd accounting
Kueue (CNCF)	K8s-native queue	Очереди, fair-share, gang-scheduling поверх стандартного K8s Job API. Минимальный вес — не «матрешка»	Не даёт MPI, GRES, multi-tenancy в HPC-смысле
Ray on K8s	ML-фреймворк	Свой scheduler, autoscale, родной для ML-команд без HPC-наследия	Не заменяет Slurm для batch-jobs с MPI/Lustre

Когда НЕ нужен Slurm в K8s

- Чисто HPC-легаси без AI → bare-metal Slurm через Ansible/SaltStack. K8s-overlay даёт IB/Lustre overhead без выгоды
- Нужен только K8s-батч с очередями → Kueue (легче) или Volcano (если нужен gang/topology). Slurm не нужен
- ML-команда без HPC-опыта → Ray on K8s как самодостаточный стек (так делают крупные ML-провайдеры)
- Slurm в K8s нужен для on-prem «HPC + AI training + inference на одном пуле». Гибрид с внешним Slurm — slurm-bridge.
- Полный спектр закрыт: Slinky / Soperator / SUNK (Slurm-based) + Volcano / Kueue / Ray (без Slurm) + bridge (гибридные пайплайны). Задача on-prem — выбрать каркас и скомбинировать

- Универсальное железо невозможно — HPC и AI расходятся в точности, сетевом профиле и паттерне хранилища.
- Универсальная оркестрация — решена. Slurm в K8s через Slinky (upstream) или Soperator (jail, ActiveCheck) .
- Экономика решается профилем нагрузки. Оценить временной график HPC и AI; слабое перекрытие нагрузок → универсальный.
- Главный вызов — команда. Нужны компетенции MPI/Lustre/IB и K8s/PyTorch/Triton одновременно.



Спасибо!



www.e-flops.ru



+7(495) 795-33-93



info@e-flops.ru



t.me/E-flops



[Dzen.ru/e_flops](https://dzen.ru/e_flops)



[VK.com/e_flops](https://vk.com/e_flops)