Multicloud.io

THE QUEUE IS FULL · THE MARKET ISN'T

Running Slurm Where Compute Is Abundant

Abundance and price are the same fact: the market deep enough to always have capacity is the market that sets the floor. The identical machine trades **82% apart between regions**; the cheapest qualifying spot vCPU sits at roughly **a sixteenth of typical on-demand**. Multicloud chooses the machines — cloud, region, and instance type — from a live catalog across AWS, Google Cloud, and Azure, so your queue is sized by that market instead of by one reservation, and priced at its floor. Two workloads carry the argument from here to the last page: **the simulation** — 2,000 cores of molecular dynamics that must stay together in one region for weeks, **57% off and 28 days cut to six** — and **the sweep** — a nightly Monte Carlo risk run that can be anywhere at all, **~90% below on-demand**. Two doorways: spill-over capacity behind the Slurm cluster you already run, or a complete Slurm environment behind one URL and your SSO.

Slurm / HPC Whitepaper · July 2026 · multicloud.io

EXECUTIVE SUMMARY

The argument, in one page

Every Slurm site knows both halves. **The queue half:** a deadline meets a fixed-size cluster, and researchers wait days for cores sitting idle somewhere they can't reach. **The bill half:** in scientific simulation, genomics, rendering, and backtesting, compute isn't overhead — it *is* the cost of the product. And it's bought from a market whose spreads a single-cloud setup never sees. The identical machine trades **82% apart between regions**; the cheapest spot core anywhere sits at **a sixteenth of typical on-demand**; and the floor is never reliably in one cloud.

The scheduler is not the problem. Slurm is the de-facto standard of world HPC, it already knows how to create machines on demand, and it now runs natively on Kubernetes via SchedMD's own **Slinky** operator. What Slurm deliberately does not solve is the layer below: *which machines should exist, in which cloud and region, at what price* — the script that buys them calls one cloud's API, because someone at your site wrote it against one cloud's API.

> Slurm decides which job runs next. Multicloud decides which machines exist — the cheapest qualifying nodes across every region of AWS, Google Cloud, and Azure, spot-aware, risk-scored, healed when reclaimed. The two meet at a queue named *burst*.

Two doorways in, no migration either way. **Extend your cluster** (early access): your existing Slurm environment gains a spill-over queue backed by the pool — the baseline stays where it is, the peaks land on the market. Or **Slurm served directly**: a complete, isolated Slurm environment operated on the pool behind one URL and your company login — a web portal for humans, an API for pipelines — assembled per engagement as a design-partner program.

Where machines should sit depends on the work, so the paper argues through two workloads it never lets go of. **The simulation** — a lab's 2,000-core molecular-dynamics run — must stay in lockstep, so it pins to **one region chosen by live price and capacity**: worth 45–61% before spot even enters. **The sweep** — a risk desk's nightly Monte Carlo, 500,000 core-hours a month — is thousands of tasks that never speak to each other, so it roams the whole board: **~90% below on-demand**, spread across markets so no single reclaim wave can stop it. A third axis compounds both. Nobody actually wants cores; they want arithmetic, and the two aren't the same market — the catalog prices every machine by **measured compute**, and the cheapest core is rarely the cheapest work. That's another 31–37% inside the already-cheapest region. And because the bill is machine-hours × rate, fleet size cancels out: savings convert into calendar at will. The simulation runs 28 days for $45.7k as typically bought, or six days for $19.6k on the pool. Cloudflare R2 holds the data somewhere neutral, so it costs nothing to read it into any cloud — and data stops dictating where compute happens.

Claims stay falsifiable: every number is a dated catalog pull or a cited source, every capability carries its status, and the comparison is measurable on your own workloads — read-only — before anything changes.

CONTENTS

What's in this paper

This paper describes a reference architecture on the platform as it stands, with status flags where a capability is **(early access)** or assembled per engagement. Slurm, Slinky, and Open OnDemand are upstream open-source projects, used unmodified; Cloudflare R2 is a third-party service. Prices are dated live-catalog pulls and public list prices; ask us to run the live catalog — or the Advisor, read-only on your own environment — for current numbers on your workloads.

SECTION 01

01 The problem: a queue problem first, a bill problem second

The capacity you need exists. Your queue just can't reach it — and reach is also price, because the pool deep enough to absorb your peak is the same pool that holds the floor. In simulation, genomics, rendering, and backtesting, compute isn't overhead — it *is* the cost of the product. Three facts about the market it's bought from:

+82%

The identical AWS *c6g.8xlarge* listed at $0.68/hr in Mumbai and $1.24/hr in Frankfurt. Same silicon, same SKU, same everything — except the postcode (live catalog, 2026-07-22).

÷16

Spot is a cloud's unsold inventory: same hardware, sold cheap, reclaimable at short notice. The cheapest one meeting real constraints that day: $0.0021 per core-hour.

+61–79%

The fifteen cheapest spot cores anywhere spanned ten regions on four continents. The floor sat in Azure that day; a team locked to either other cloud paid 61–79% more — and tomorrow it sits somewhere else.

The queue half. A deadline meets a fixed-size cluster; the queue backs up; researchers wait days for cores that are sitting idle somewhere they can't reach. Cloud bursting was invented for exactly this — and every implementation of it stops at one cloud.

The bill half. Every vendor ships a respectable way to run Slurm *on that vendor's cloud* — and each one quietly manages you deeper into it. Slurm's own scaling hooks have the same horizon: the script that creates nodes calls a single cloud's API, because someone at your site wrote it against a single cloud's API. Nothing in the standard toolbox answers the question that actually moves the bill: *which cloud and region should this batch land on today?*

> This paper fixes the reach, not the scheduler: Slurm unmodified — including SchedMD's own Kubernetes operator — on a capacity layer that continuously buys the cheapest qualifying nodes across every region of AWS, Google Cloud, and Azure, with Cloudflare R2 as a cloud-neutral data plane. Two doorways in; nothing migrates either way.

THE TWO WORKLOADS

Two workloads, followed to the last page

Architecture arguments are cheap in the abstract, so this paper picks two workloads now and refuses to let go of them. They sit at opposite ends of everything that matters — how much the pieces talk to each other, whether a lost machine hurts, how much data moves, what the deadline looks like — so between them they bracket most of what a Slurm site actually runs. Every section from here answers the same two questions: *what does this mean for the simulation, and what does it mean for the sweep?*

	THE SIMULATION	THE SWEEP
Who	A structural-biology lab	A quantitative-finance risk desk
The work	GROMACS molecular dynamics — one long simulation, run for weeks	Monte Carlo scenario evaluation, run nightly
Shape	*Tightly coupled*: every process talks to its neighbours, constantly	*Embarrassingly parallel*: hundreds of thousands of tasks, none aware of the others
Appetite	2,000 cores held together for weeks (~1.34M core-hours)	500,000 core-hours a month, in six-hour overnight windows
Interruption	Lose one machine, lose the job — checkpoint, or pay for certainty	Lose one machine, lose one task; Slurm re-runs it
Data	Tens of TB of trajectories out	Kilobytes in, kilobytes out, per task
The deadline	A submission date months away that everything is measured against	Tomorrow morning, before the market opens
Must run	**One region** — latency gates the science	**Anywhere** — the market decides, continuously

They want opposite things, and that is the point. The simulation can't be scattered across clouds, so it wins by landing in the right place on the right silicon — bought once, held for the run: section 8 takes it from $45.7k to $19.6k, and from 28 days to six. The sweep is scattered by nature, so it wins by contesting every market, every night: section 9 takes it from $17k a month to $1,649. One is an argument about buying well; the other is an argument about reach.

> The same layer underneath serves both, because both are the same question at different scales — *which machines should exist right now, and where?* Neither is a customer anecdote: each is a documented workload shape priced against a dated catalog pull, so every figure here can be checked, re-run, and argued with.

A note on units: this paper says *core* in prose where the price tables say *vCPU*. They are the same thing — the unit the clouds actually sell and bill. Prices are quoted per vCPU-hour because that is how they are published; nothing is converted behind your back.

SECTION 02

02 The scheduler layer is solved: Slurm — now on Kubernetes

You do not need to replace Slurm, and this paper will not ask you to. Slurm is the de-facto standard scheduler of world HPC — thirty years of queue policy your users and pipelines already speak fluently: who gets machines first, who has used more than their share, which jobs can slip into the gaps. Two upstream developments matter here.

Slurm already knows how to burst — one cloud at a time

Slurm can define nodes that don't exist yet. When work needs them, it runs a script your site supplies to create the machines; when they sit idle, another script hands them back. This is the mechanism underneath every cloud vendor's Slurm tooling — and the catch is in the words *your site supplies*: that script talks to one cloud's API, with one cloud's credentials, images, and networking. A second cloud means writing, operating, and debugging a second script — and the *choice* between the two is a human reading price lists.

Slurm now runs natively on Kubernetes — per its own maintainers

SchedMD's **Slinky** project ships an official operator that runs Slurm's own daemons — the controller, the accounting database, the REST endpoint, the worker agents — as ordinary Kubernetes pods, with worker pools that scale, drain, and shrink to zero. It reached v1.0 in late 2025 — and in December 2025 **NVIDIA acquired SchedMD outright**, pledging continued open-source, vendor-neutral Slurm development; NVIDIA reports running Slinky in production at 8,000-plus GPUs. Nebius's open-source *Soperator* proves the same shape independently — usefully keeping the operator layer a two-source market. The direction is no longer controversial: Slurm's queue semantics on top, Kubernetes' machine lifecycle underneath.

> That convergence is the hinge of this paper. Once a Slurm worker is just a pod, "where do Slurm's nodes come from?" becomes a Kubernetes question — and that is a question a cross-cloud capacity layer can answer.

One non-answer, for completeness: Slurm **federation** lets sibling clusters inside one organization share a queue. It answers "which of my clusters runs this job?", not "which market builds my cluster today?" They compose; they don't compete.

SECTION 03

03 What Slurm assumes — the seam

Read Slurm's documentation closely and a boundary appears. Slurm decides **which job runs when, on the machines it has been told about**. Everything below that line is somebody else's problem — which today means yours:

SLURM OPTIMIZES	SLURM ASSUMES SOMEONE ELSE PROVIDES
Which job runs next — priority, fair-share, gap-filling	Which **machines** exist, and where
Queues, limits, accounting	Which **cloud, region, and price** each machine has
Job dependencies and arrays	The script that buys them — one per cloud, yours to write
Handing machines to jobs	Spot reclaims: noticing, draining, replacing
—	Cross-cloud networking, images, credentials, quotas

This is the seam, and it is where the money is. Slurm can be brilliant with the machines it has; if those machines were bought in the wrong region at the wrong price — or capped by one region's quota while a deadline burns — the queue is being optimized on top of a bill that was already lost. Our two workloads hit this seam from opposite sides: the simulation needs one right region, chosen once and held for weeks; the sweep needs the right markets, chosen again every night. Slurm answers neither — on purpose.

> The regional spreads and the spot floor of section 01 live **below** the scheduler, in the layer that decides what to buy. That layer is what Multicloud is.

That is not a flaw in Slurm. It is the layering that has let Slurm outlive every hardware generation it has ever scheduled: the scheduler works in seconds against the fleet it has, while buying machines works in minutes against a moving market. The rest of this paper is about running the first on the second.

SECTION 04

04 The capacity layer: one decision over every cloud

Multicloud operates the capacity market the scheduler assumes. The mechanism, briefly — the full story is the companion *Architecture Whitepaper*.

A live catalog. Roughly **265,000 priced offers** across ~3,900 machine types — every region and zone of AWS, Google Cloud, and Azure, spot and on-demand, each carrying how likely it is to be reclaimed and **how fast its actual silicon measures**. That last part matters more in HPC than anywhere: "32 cores" is not a unit of work, and the catalog knows which 32 cores do more arithmetic per dollar.

One placement decision. The scheduler adds up what's waiting and buys the **cheapest machines that qualify** — from anywhere in that catalog. *Qualify* means whatever you declared: enough memory, fast enough silicon, a GPU class, allowed locations, a spend cap. Spot where your constraints tolerate interruption, on-demand where they don't. The simulation and the sweep differ *only* in what they declare — same decision, same catalog, opposite answers.

Interruptions as weather. Every machine runs an agent listening for its cloud's reclaim warning. On notice: drain the work, buy the replacement from the catalog — often in another zone or region entirely — and have it serving in about a minute. Machines are replaced, never repaired; none of them holds anything you can't lose.

One network, one cluster. Machines in different clouds join a single encrypted network under one standard Kubernetes control plane. A cluster spanning AWS, Google Cloud, and Azure is still *one cluster* — one config file, one scheduler's view, no federation project to run. Each customer gets their own: own control plane, own private networks in every cloud it touches, own certificate authority, nothing shared with anyone.

Your accounts, your bill. Machines are provisioned into dedicated cloud accounts you own. Your negotiated discounts, committed-use plans, and credits still apply; Multicloud charges for the platform, never a markup on machines.

> Like a power plant behind a wall socket, the machinery is ours to run and what reaches you is standard. For a Slurm shop the socket has two sides: ordinary Kubernetes for your platform team — where SchedMD's own charts **just install** — and ordinary Slurm for your users, who go on submitting jobs exactly as they did and never learn a single word from this section.

SECTION 05

05 Doorway one: extend the cluster you already run (early access)

Nothing migrates. For a site already running Slurm, the entry point is a **burst partition** — a queue named *burst*. Your existing machines keep carrying the baseline exactly as configured today; the new queue maps to capacity the pool buys on demand and hands back when idle. Users opt in the way they already express everything else, by naming the queue when they submit, and your policy still governs: limits, priorities, fair-share, and accounting all apply to burst jobs like any others. Never beg for quota or wait for capacity. Never overpay for it. And never migrate to get there.

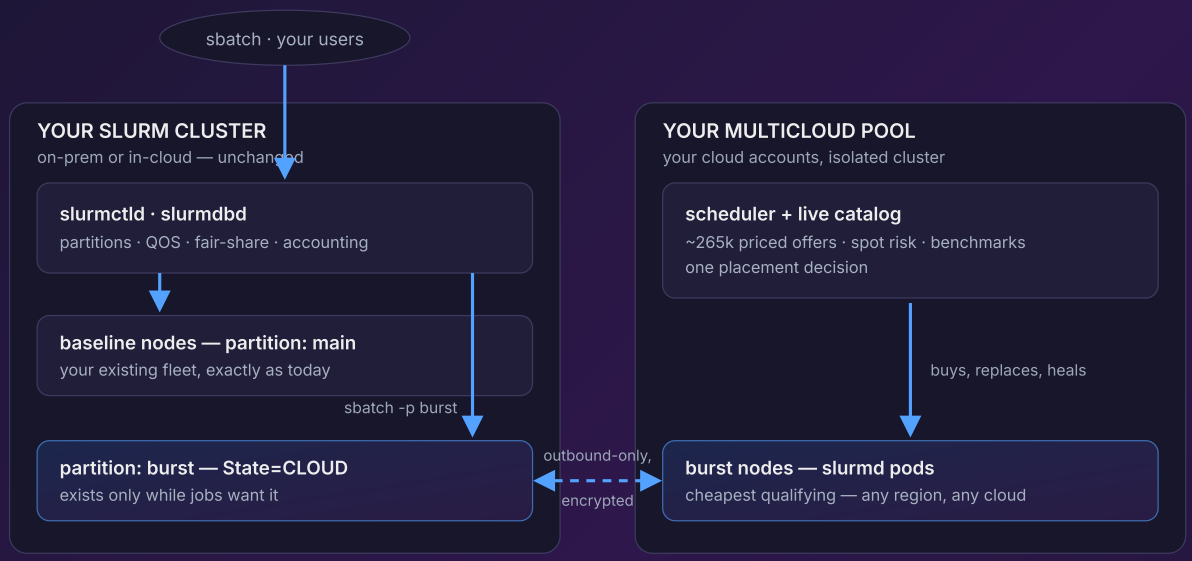

Figure 1 — the burst partition: Slurm's queue, policy, and accounting stay home; pending burst demand drives the pool, which buys the cheapest qualifying nodes on the whole board and hands them back when idle.

Two wirings, depending on where your control plane lives

Slurm on Kubernetes (Slinky / Soperator). Your cluster gains one **virtual node** backed by the pool — a node that isn't a machine, but a doorway to thousands. It is fenced off, so nothing lands on it by accident; point your burst workers at it and they run on the pool while still showing up in your dashboards and your queue listings exactly like local ones. The connection is outbound-only and encrypted. No foreign machine ever joins your cluster.

Classic Slurm (bare metal or VMs). Your burst nodes are the same create-them-when-needed nodes Slurm has always supported — except the script asks the pool for capacity instead of calling one cloud's API. Same five-line contract, pointed at every cloud at once. The workers register with your controller over the encrypted network, and hand the machines back when the queue goes quiet.

```
# slurm.conf — the burst partition, classic wiring
NodeName=mc-burst-[001-256] State=CLOUD CPUs=32 RealMemory=63000 Feature=multicloud
PartitionName=burst Nodes=mc-burst-[001-256] State=UP MaxTime=24:00:00
ResumeProgram=/opt/multicloud/resume.sh     # asks the pool, not one cloud
SuspendProgram=/opt/multicloud/suspend.sh
SuspendTime=600
```

Only one thing changes, and it changes at exactly the moment your cluster fills up. Instead of the queue deepening behind fixed capacity — or behind one cloud's quota — the waiting work becomes a buy signal. The pool takes the cheapest qualifying machines on the whole board, joins them in about a minute each, and releases them when they go idle.

The sweep goes through this doorway first, and that is the reason to start there: a nightly, interruption-tolerant, re-runnable queue is the lowest-stakes thing a site owns. Point one partition at the pool, leave everything else exactly where it is, and the delta shows up in your own accounting within a week. The simulation can follow through the same door once the burst partition has proven itself — with *--constraint* pinning it to a single region (section 07).

> Spill-over is the doorway, not the ceiling: the same partition mechanism runs from "5% overflow at deadline week" to "the baseline is the small half," one *slurm.conf* edit at a time, with no migration event anywhere along the way.

SECTION 06

06 Doorway two: Slurm as a front door — one URL, your SSO

(design-partner program)

The second doorway is for teams with no cluster to extend — or no appetite to keep operating one. Multicloud stands up a **complete Slurm environment on the pool** and serves it the way modern software is served: a public URL, a padlock, and your own company login. This is the doorway for **the simulation's** lab, whose scarcest resource is graduate-student time, not core-hours, and which should never have to staff a cluster administrator to run a simulation. A reference architecture assembled from standard parts, run for you as a design-partner engagement.

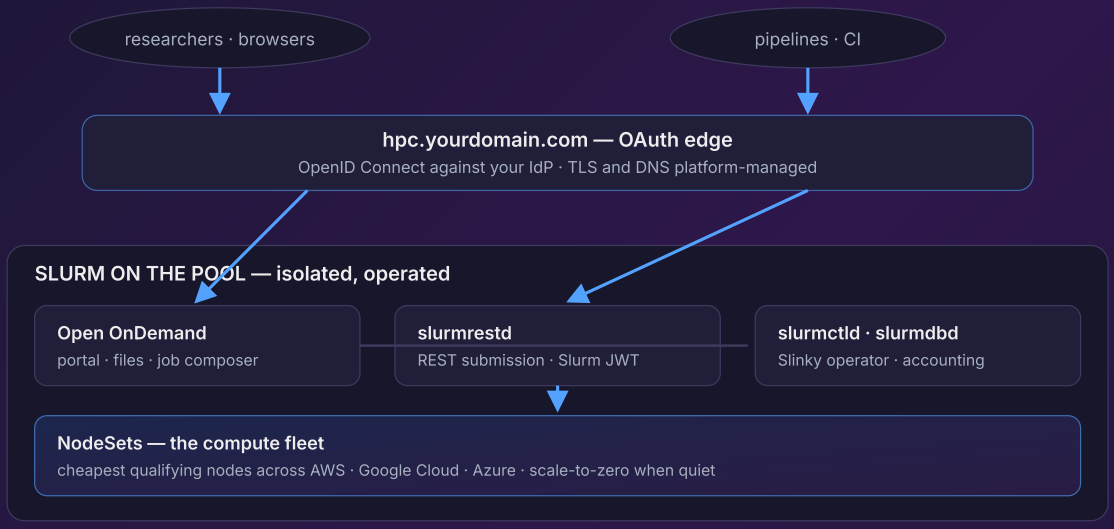

Figure 2 — the front door: one URL, OIDC against your identity provider, Open OnDemand for humans and slurmrestd for machines — and a compute fleet that exists only while the queue wants it.

The four standard parts

The cluster. An isolated, standard Kubernetes cluster built from cheapest-qualifying capacity, with SchedMD's own operator running Slurm's controller, accounting database, REST endpoint, and login sessions. Sized by demand; costs nothing when nobody is using it.

The door for humans: Open OnDemand. The standard open-source HPC web portal, from the Ohio Supercomputer Center — file management, job submission, interactive shells — with first-class single-sign-on support. Users go to *hpc.yourdomain.com*, log in with the company account they already have, and are submitting jobs in a browser. No VPN, no jump host, no SSH keys pasted into a wiki.

The door for pipelines. Slurm's REST API behind the same login, for CI systems and scripted submission. The gateway checks the caller's identity and swaps it for the credential Slurm expects, which is the standard way to front it.

The edge. DNS, certificates and their renewal, and the login proxy are the platform's own serving machinery — the same parts that front Multicloud's own surfaces, assembled for your tenancy as part of the engagement.

The pitch here is Simplicity at HPC scale: a research group gets what a national lab's user services team provides — portal, single sign-on, accounting, a queue with real capacity behind it — without hiring anyone to run the platform underneath.

> And because the substance is plain Slurm on plain Kubernetes in accounts you own, the door swings both ways: ask for the keys and operate it yourself whenever you like. The Simplicity is optional, and it doesn't lock.

SECTION 07

07 Two shapes of HPC work, two placement answers

Everything so far bought cheap machines. *Where* those machines should sit is a property of the work itself, and HPC has two classic shapes with opposite answers — the two the simulation and the sweep were picked to represent. The platform expresses both with the same two knobs: how tightly a job's machines must stay together, and which places are allowed at all.

Shape one: tightly coupled — the simulation. Molecular dynamics, fluid dynamics, weather: one model split across many machines that must stay in lockstep, exchanging results every step. The whole job runs at the speed of the slowest conversation, so the network between machines decides how far it scales. That ladder is steep — specialist supercomputer fabrics answer in well under a microsecond, the clouds' best networking sits around 5–10 µs, and ordinary networking is slower again. An encrypted link *between* clouds, across the public internet, is a terrible place to run this, and this paper will not pretend otherwise. The honest answer is **every machine in one region** — often one building. The cross-cloud advantage doesn't disappear; it moves up a level, from the *packets* to the *purchase*: the pool picks the region the way it picks everything — the cheapest one, anywhere, with the capacity to hold the job — then keeps every machine inside it.

Shape two: embarrassingly parallel — the sweep. Monte Carlo runs, genomics pipelines, rendering frames, parameter studies: thousands of independent tasks that never talk to each other. (The name is a genuine term of art, and slightly unfair — this is the easy kind of parallel.) No grouping, no location rules. Every machine is bought on its own merits, wherever qualifying capacity is cheapest *right now*. Interruption is cheap by nature: lose a machine, lose one task's progress, and Slurm simply runs it again.

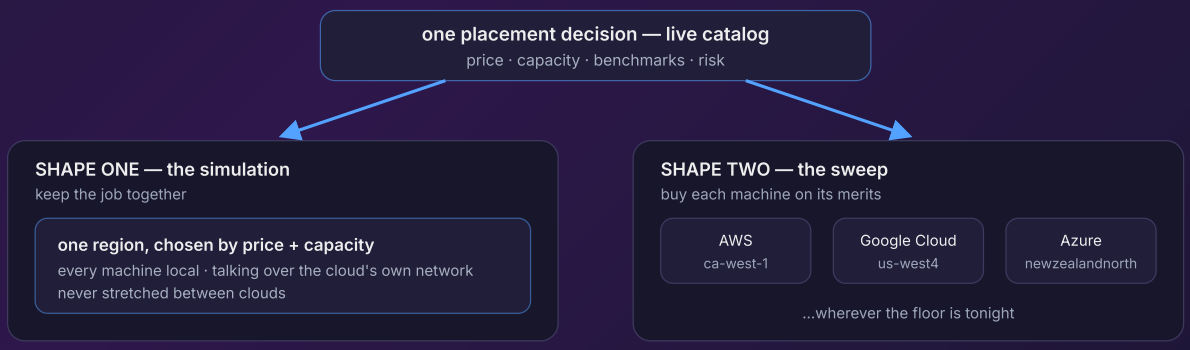

Figure 3 — two shapes, two answers: the tightly coupled job gets a region chosen by the market and stays inside it; the parallel sweep contests every purchase across the whole board.

One cluster can hold both shapes at once — a region-pinned queue for the simulation, a roaming scavenger queue for the sweep — because the rules attach to groups of machines, not to the cluster. Your users see two queue names. Underneath them sit two different answers to "where should machines exist?"

SECTION 08

08 The simulation, priced: coupled, pinned to one region

The first workload, in full. GROMACS is the workhorse of molecular dynamics, and it does scale — a 2025 peer-reviewed study drove it to 65,536 cores at better than 90% efficiency, though on a purpose-built supercomputer fabric (Kutzner et al., *J. Comput. Chem.*). The lab wants **2,000 cores held together for a multi-week run**: 63 machines of 32 cores each, all in one region, talking over the cloud's own network. So the placement question is a single line — *which region?* — and it is worth real money:

CLOUD / REGION	INSTANCE	$/VCPU-HR	2,000 VCPU × 1 HR
AWS ap-south-1 (Mumbai) — chosen	c6g.8xlarge	$0.0213	**$42.60**
Google Cloud us-central1	n4a-highcpu-32	$0.0325	$65.00
AWS us-east-1	c6g.8xlarge	$0.0340	$68.00
Azure eastus	Standard_D32ps_v6	$0.0351	$70.20
AWS eu-central-1 (Frankfurt)	c6g.8xlarge	$0.0388	$77.60
Azure westeurope	Standard_E32ps_v6	$0.0551	$110.20

Cheapest qualifying on-demand machine per sampled region, live catalog, 2026-07-22. Best to worst is a **61% spread** for the same hour of the same-sized cluster; even holding the machine type fixed — AWS *c6g.8xlarge*, identical hardware — Mumbai is **45% cheaper than Frankfurt**. Across a four-week run the chosen region bills about **$28.6k**, the worst row **$74.1k**. No grant proposal in history has a line item for that.

Spot is a dial here, not a default. A tightly coupled job dies with its slowest machine, so putting all 63 on reclaimable capacity multiplies the risk. The catalog scores that risk per zone, so the choice is explicit rather than hopeful: runs that can checkpoint ride low-risk spot and restart when unlucky; runs that can't run on-demand and take the regional spread — which is what the 45–61% above already is, before any spot at all.

Processor architecture is a constraint you declare. The cheapest rows here are ARM chips. GROMACS builds cleanly on ARM; code that doesn't is one constraint away from an Intel-and-AMD-only view of the catalog, at a somewhat higher floor. And these are point-in-time prices: the *existence* of the spread is durable, its *layout* moves — which is the argument for deciding by live query instead of by whichever region a spreadsheet picked two years ago.

Data doesn't complicate the choice, because the simulation's storage is cloud-neutral: inputs stage in from Cloudflare R2 at no data cost into any region, and results write out once and are readable everywhere afterwards (section 10). The region decision stays a pure compute-price decision, which is exactly what it should be.

Buying arithmetic, not cores — the third arbitrage axis

Everything above priced *cores*. But nobody wants cores; they want arithmetic, and the two are not the same market. A core is a unit of rental. Arithmetic is the unit of work. The catalog measures how fast each processor model actually is and prices every offer per unit of measured compute — and that ranking is not the price-per-core ranking. Same region, same day, same 32-core size (AWS ap-south-1, on-demand, 2026-07-22):

INSTANCE	$/HR	MULTI-CORE BENCH	$/VCPU-HR	$/1K-BENCH-PT-HR
c6g.8xlarge (Graviton2) — cheapest per vCPU	$0.682	11,616	**$0.0213**	$0.0587
c6a.8xlarge (AMD Milan)	$0.748	8,609	$0.0234	$0.0869
c7g.8xlarge (Graviton3)	$0.785	17,683	$0.0245	$0.0444
c8g.8xlarge (Graviton4) — cheapest per benchmark	$0.864	21,502	$0.0270	**$0.0402**
c6i.8xlarge (Intel Ice Lake)	$1.360	13,791	$0.0425	$0.0986

The previous page's winner, *c6g.8xlarge*, is the genuine price-per-core champion — and a trap. Per unit of arithmetic delivered, *c8g.8xlarge* is **31.5% cheaper**, despite costing 27% more per core-hour. Shopping by the obvious number overpays by **46%**. Not a one-region quirk: the same swap is worth 36.6% in us-east-1 and 36.9% in eu-central-1, where the obvious number overpays by 58%. Re-ranking by arithmetic leaves the *region* order untouched — Mumbai still wins — but changes the winning *machine* inside every AWS region.

> Restated in the currency the lab actually cares about, the four-week run's **$28.6k becomes about $19.6k: another 31.5% off a bill that was already the best region's**. And the same work now arrives on ~34 machines instead of 63 — fewer participants, less chatter between them, a straight win for a job whose speed is set by its slowest conversation. (At a fixed 2,000 cores, the saving shows up as calendar instead.)

Work that runs on one core at a time ranks differently. Serial stages, and codes licensed per core (chip design is the classic), are buying single-core speed rather than throughput — and the contenders diverge sharply by shape. Picking the wrong one costs 14.5–39.3% in the sampled region. Both kinds of speed are first-class constraints: declare the shape of your work, and the catalog query does the rest.

Honesty notes: these scores are the catalog's own index, measured per processor model — a signal for picking machines, not a benchmark of *your* code, and only directional when comparing ARM against x86. Run your own kernel before committing a run. Coverage grows but isn't total: Azure's newest ARM line has scores for its smaller sizes in this pull (section 09 uses one), while its largest sizes — including two rows in the previous page's regional table — are still awaiting samples. Unscored machines are never guessed at; they compete on price and size alone.

SECTION 09

09 The sweep, priced: parallel, pinned to nothing

The second workload, in full. The risk desk's nightly run is hundreds of thousands of independent scenarios, needing no checkpoints because a lost task is simply run again: **500,000 core-hours a month**, kilobytes in and out per task. Where the simulation chose one market and committed, the sweep contests all of them — every machine bought on its own merits:

CLOUD	REGION	INSTANCE	$/VCPU-HR	RISK BAND
Azure	newzealandnorth	Standard_F16ams_v6	**$0.00208**	0–5%
Google Cloud	us-west4	e2-highcpu-8	$0.00336	0–5%
Google Cloud	northamerica-northeast2	n2-highcpu-8	$0.00356	0–5%
AWS	ca-west-1	c6g.2xlarge	$0.00373	10–15%
Azure	centralindia	Standard_D8pls_v6	$0.00377	>20%
AWS	us-east-2	c6gd.2xlarge	$0.00384	>20%
Google Cloud	asia-south2	n2-highcpu-8	$0.00388	0–5%

Cheapest qualifying spot capacity anywhere, re-ranked by price per vCPU (live catalog, 2026-07-22). Read it twice. The floor sits in a region most teams have never priced in their lives; the fifteen cheapest rows span AWS, Google Cloud, and Azure across ten regions on four continents; and price rank does not track risk rank — two rows priced within a whisker of each other differ fourfold in how likely they are to be taken back. So a fleet should *spread across* these rows rather than pile onto rank #1 — precisely what a single-cloud setup cannot do.

SCENARIO — 500,000 VCPU-HR/MONTH	$/VCPU-HR	MONTHLY	VS. ON-DEMAND
(a) Single-cloud on-demand (AWS us-east-1, c6g)	$0.0340	$17,000	—
(b) Single-cloud spot — best AWS-only row	$0.00373	$1,863	−89.0%
(c) Cross-cloud spot — top-5 blend	$0.00330	**$1,649**	**−90.3%**

Honesty note on row (c): two of the five blended rows are Azure — priced live, provisioning in final validation. Restricted to AWS and Google Cloud today: $1,836 a month, −89.2%; the Oceania and India floors join as Azure validation completes.

The honest reading: the big jump is on-demand → spot, and any single cloud gives you most of it — *if* your team builds and operates the reclaim handling itself. Going cross-cloud is the last **11.5% off the best single-cloud spot bill**, plus what the table can't show: when one cloud's spot market tightens, a single-cloud fleet stalls, while a fleet with somewhere else to go re-lands and keeps running. That handling is the pool's job, not a script your team maintains per cloud.

And section 08's arithmetic lens bites harder here, because a sweep is pure throughput. Per unit of compute the same #1 wins, then the board re-sorts *both ways*: Azure's newest ARM row drops from +81% on sticker to **+40%** — fresh silicon the price-per-core lens mispriced in the buyer's favour — while the legacy machines' mild-looking 61–79% premiums balloon to **148–279%**. Shopping by measured compute buys 2.5–4× more work per dollar.

The whole-workload ledger: cheaper and faster are the same knob

Budgets get approved as totals and deadlines as dates, so here are both workloads restated whole. One property of an elastic pool deserves its own line first: **the bill is machine-hours × rate, so the size of the fleet cancels out**. Double the machines and the days halve; the bill doesn't move. Time stops being a consequence and becomes a knob — which is why each table carries a clock column next to the bill column, and why the fastest rows and the cheapest rows turn out to be the same rows.

The simulation (section 08) — a fixed amount of arithmetic to get through. "Default" is what most teams do: a US region, cores as the unit.

STRATEGY	FLEET	WALL-CLOCK	TOTAL BILL
Default: c6g, us-east-1, on-demand	~63 nodes	28 days	$45.7k
+ region arbitrage — c6g, ap-south-1	~100 nodes	17.5 days	**$28.6k (−37%)**
+ benchmark arbitrage — c8g, ap-south-1	~79 nodes	12 days	**$19.6k (−57%)**
+ deadline-first — c8g, ap-south-1	~158 nodes	**6 days**	**$19.6k (−57%)**

Read the last two rows together: doubling the fleet halved the calendar and left the bill untouched. Fleet sizes here are choices, not consequences — bounded only by how far the code scales (the GROMACS study cited earlier holds above 90% efficiency to 512 machines; the largest row above is 158). The worst sampled region ran the same run at $74.1k.

The sweep (section 09) — 500,000 core-hours a month; call the overnight window six hours, which today's on-demand fleet just fills.

STRATEGY	NIGHTLY FLEET	WINDOW	MONTHLY BILL
Today: on-demand, one region	~2,800 vCPU	6 h	$17,000
Cross-cloud spot, same fleet	~2,800 vCPU	6 h	**$1,649 (−90%)**
Cross-cloud spot, 10× fleet	~28,000 vCPU	**~36 min**	**$1,649**
Same bill as today, 10× the scenarios	~28,000 vCPU	6 h	≈$16.5k — **10× the science**

The symmetry is the point: the same science for a tenth of the money, or ten times the science for the same money, and the clock comes free either way. Caveats stay on the table. A fleet ten times the size needs ten times the available capacity — which is exactly what spreading across the markets on the previous page buys you. Interruptions requeue, which this shape of work shrugs off. And every figure is the 2026-07-22 snapshot, recomputable live.

SECTION 10

10 The data plane: Cloudflare R2, the end of data gravity

The quiet force pinning HPC to one cloud isn't compute. It's data. Clouds charge nothing to put data in and roughly **$0.09–0.12 per GB** to take it out again, so 30 TB of inputs sitting in one cloud's storage means every burst somewhere else opens with a **$2,700–3,600 invoice — every time**. Results are worse: they pile up *inside* whichever cloud ran the job and leave at the same rates. This is data gravity, and it quietly cancels every bit of placement freedom the sections above just bought. It is also not an accident of physics.

So this architecture keeps the data somewhere neutral: **Cloudflare R2**, which speaks the same storage API everyone already uses and charges **nothing to read data out, at any volume** (R2 pricing, checked 2026-07-22):

	CLOUDFLARE R2	TYPICAL HYPERSCALER OBJECT STORE
Storage (standard)	$0.015 /GB-month	~$0.02–0.023 /GB-month
Reads (class B)	$0.36 /million	~$0.40 /million
Writes (class A)	$4.50 /million	~$5 /million
Egress to internet	**$0 — always**	**~$0.09–0.12 /GB** (per GiB on Google Cloud) after ~100 GB free
Egress to another cloud's compute	**$0**	same as internet egress

Your tooling doesn't change — same commands, same libraries, one endpoint URL swapped. The pattern that fits HPC is **pull, compute, push**: copy inputs to the machine's local disk when the job starts, work against local disk, write results back at the end. This is a library, not a shared filesystem — scratch space stays local, and writes favour fewer, larger files so the per-operation charges stay in the noise.

> The honest asymmetry: writing results *out of* the cloud that produced them still pays that cloud's exit fee once — neutral storage can't waive another company's toll. The economics rest on paying it **once**, on results (usually a fraction of the input volume), instead of paying it again and again on inputs and re-reads. The repeat charges are the trap, and they go to zero.

What zero egress changes, per shape

Figure 4 — the cloud-neutral data plane: inputs stage into any cloud free (R2 charges nothing outbound, clouds charge nothing inbound); results pay the producing cloud's egress once and are then free to every reader, everywhere, forever.

The simulation (tens of TB of results). Inputs land in *any* region of *any* cloud for nothing — the store charges nothing to send, the clouds charge nothing to receive. Where last year's data happens to live no longer constrains where this year's run runs.

The sweep (kilobytes per task). A fleet scattered across ten regions reads the same scenario library, freely. There is no "wrong cloud" for a task to land in — which is exactly what lets every machine be bought on its own merits.

Both. Results stay readable forever — by the next stage of the pipeline in a different cloud, by analysis back at the lab, by a collaborator's laptop — with no exit toll on top. Your data stops being a hostage of wherever it happened to be computed.

Caveats in the open: this store keeps no version history (put provenance in your naming or a manifest), its cold tier charges for retrieval, and its per-operation pricing punishes millions of tiny files. None of that bites the pull-compute-push pattern; all of it would bite anyone who tries to mount it like a filesystem.

SECTION 11

11 Why this stack, on this pool

Each advantage here is structural — a consequence of where the platform sits, not a bolt-on.

1 · One integration instead of one per cloud. Multi-cloud Slurm today means a separate provisioning toolchain for every cloud — scripts, images, permissions, quotas — plus a human deciding between them. Here the whole board answers one signal, and the decision is a catalog query.

2 · The whole board, in one decision. Per-cloud tooling optimizes *inside* the cloud it belongs to. Nothing in that toolbox prices the choice section 08 showed was worth 45–61% before spot even enters: which cloud and region the cluster should exist in at all.

3 · Spot survival is built in. Noticing the reclaim, draining, requeuing, replacing — risk-scored when buying, healed when losing. The very burden that keeps most HPC fleets paying on-demand prices is the part that arrives as platform, not as a per-site build.

4 · Arithmetic, not core counts. HPC buys work done, so the catalog prices machines by measured compute and treats speed floors as first-class constraints. "Cheapest that qualifies" therefore means cheapest *per unit of work*, not per marketing core. In this paper's own tables: another 31–37% inside the already-cheapest region (section 08), and spot picks priced 148–279% above the true floor exposed (section 09).

5 · Every customer gets their own cluster. Own control plane, own private networks, own certificate authority, own encrypted fabric. For the pharma, chip-design, and defense-adjacent work that dominates commercial HPC, that isolation is a structural given rather than a compliance addendum.

6 · Your accounts, no markup, both doors open. Machines at cost in accounts you own; unmodified open-source Slurm on top; conformant Kubernetes underneath. Adopt it as a service, walk away with a working cluster — the exit is part of the product.

The landscape, honestly — named on the next page. Running Slurm in the cloud is a solved problem many times over: every major cloud ships it managed, the GPU clouds bundle it, portals resell it, brokers price-shop it per job. The next page puts the field in one table. The seam that stays empty across all of it: an **operated pool on your own accounts, spanning AWS, Google Cloud, and Azure, with the machine-picking automated and Slurm unmodified on top.**

The field, by name — and the column that stays empty

ROUTE	SLURM SURVIVES?	WHO PICKS CLOUD · REGION · MACHINE	SPOT LIFECYCLE	WHAT THE TOOL COSTS (COMPUTE ALWAYS EXTRA)
Single-cloud builders — ParallelCluster · CycleCloud · Cluster Toolkit	Yes — you operate it	You · you · you, in config files — one cloud	Requeue hooks; policy yours	Free tools + the team that runs them
Managed single-cloud Slurm — AWS PCS · Azure CCWS · Google Cluster Director	Yes — vendor-run control plane	Fixed · you · you, as static node groups	Requeue; healing varies	PCS: ctrl $0.58–6.54/hr + **$0.08/inst-hr**; CCWS & Cluster Director free
GPU clouds, Slurm included — CoreWeave SUNK · Nebius · Lambda · Crusoe	Yes — theirs, managed	Their fleet · their DCs · contract	Reserved terms; no spot market	Bundled into committed $/GPU-hr
HPC SaaS portals — e.g. Rescale + its Slurm connector	Portal-mediated	Multi-cloud, from their offer sheet, on their platform	Varies	Their core-hour rates, commitment-priced; list not public
Spot-rescue overlays — MemVerge · Exostellar	Yes — untouched	Wherever you already are, per cloud	Checkpoint / live-migrate off reclaims	Commercial license, vendor-quoted
Client-side job brokers — SkyPilot	No — replaces the front end	Per job: cloud + region + SKU, price-aware, your accounts	Job-level auto-recovery	OSS free, you operate it; platform pricing undisclosed
This paper's stack	**Yes — unmodified, either doorway**	**The platform: live catalog over AWS · Google Cloud · Azure, benchmark floors, risk scores — your accounts**	**Detect · drain · requeue · replace, operated**	**Flat fee under audited savings; no markup**

Read the third column all the way down: in every row but the last, *which cloud, which region, which machine* is a human being or a config file. **Everywhere else on this table the list of machines is an input. Here it is an output.** Credit where due: SkyPilot genuinely price-shops per job (but replaces the front end, and prices cores, not arithmetic); GPU-cloud contract rates often beat hyperscaler GPU list, so price them for training; Rescale's licensed-solver catalogue is value this stack does not replicate. Prices: published list where public, model shape where not (2026-07-22).

What the meters and the walls do to section 09's sweep. Take PCS's meter from the table (published rates, us-east-1, 2026-07-22). The sweep's cheapest AWS row is a small machine at about $0.030/hr — so the $0.08-per-machine-hour fee is **2.7 times the price of the thing it manages**. Across the month that's roughly $7,300 of fees on $1,863 of metered compute, where the cross-cloud pool bills $1,649 all in. To be fair: on section 08's big on-demand machines the same meter fades to a defensible 9–12%. It is shaped for big-node HPC and lands hardest on scavenger fleets. Azure's and Google Cloud's tools charge **no fee at all** — their cost is the wall. One cloud's slice of the board prices at the floor only when your cloud happens to hold it, and 61–79% above when it doesn't; the floor rotates, and every actual row-picking decision stays in your team's spreadsheet.

SECTION 12

12 Adopting it: Audit → Extend → Serve

The same three phases as every Multicloud engagement, each with standalone payoff, mapped to an HPC estate:

1. **Audit** *(read-only, days, zero risk).* Point the Advisor at your current environment. It computes the counterfactual no single-cloud tool can: what your actual job mix *would* cost on the optimal region, cloud, instance family, and pricing model — benchmark-normalized, egress-aware. It measures; it changes nothing. What sections 08 and 09 did for the simulation and the sweep, it does for whatever you actually run.

2. **Extend** *(early access — start with one partition).* Attach the burst partition behind your existing Slurm cluster (section 05). Pick the lowest-risk queue you have — the overnight sweep, the student partition, the deadline overflow — and watch cost and reclaim behavior in your own accounting. The dial runs from 5% overflow to most of the fleet, one *slurm.conf* edit at a time.

3. **Serve** *(design-partner program).* Stand up the front door of section 06 — portal, SSO, REST — for the teams that should never have to think about the machinery. When you want the keys, the pool underneath is a standard cluster in your accounts: promotion to self-operated is a configuration step, not a migration.

> **The savings pay for it.** A flat platform fee, priced under your audited savings — never per-node, never a percentage of savings, never a markup on machines. The audit comes first precisely so the fee is justified by a number you measured yourself.

Where the two workloads end up

	THE SIMULATION	THE SWEEP
Placement	Affinity: cloud + region — one market, chosen by live price and capacity, held for the run	Affinity: none — every node purchase contests the whole board, re-decided continuously
Pricing model	On-demand, or low-risk spot where it can checkpoint	Spot, diversified across markets; reclaims requeue
The bill	$45.7k → **$19.6k** (−57%): region arbitrage, then benchmark arbitrage	$17,000/mo → **$1,649/mo** (−90%): on-demand → cross-cloud spot blend
The calendar	28 days → **6 days** at the same $19.6k — fleet size is a knob, not a consequence	6 h → **~36 min** at 10× fleet, same bill — or 10× the scenarios for today's money
Doorway	Served front door (section 06) — the lab runs science, not a cluster	Extend your cluster (section 05) — one burst partition, everything else untouched
Needed from the platform	The *right* market, bought once	*Every* market, bought again every night

Read across the last two columns and the paper's claim is visible in one line: the simulation and the sweep want opposite things from a capacity layer, and neither is served by picking a cloud in advance. One needs the best single market that exists on the day it starts; the other needs all of them, continuously.

> The layer that can answer both questions is the same layer — and it is the one Slurm has always assumed someone else would build.

Whichever one you recognise, start the same way

If you run something like the sweep — an overnight queue, a rendering farm, a genomics pipeline — it is the cheapest thing you own to prove. The Audit prices it read-only in days; one burst partition then tests the number for real, with your baseline untouched.

If you run something like the simulation — one long coupled job with a date attached — the Audit re-prices your *next* run against every region on the board before you commit a single node-hour, and tells you what the calendar costs at each fleet size.

STATUS & SOURCES

What's built, what's early, what's reference architecture

PIECE	STATUS
Catalog, scheduler, spot-aware placement, reclaim healing, cross-cloud fabric	**In production** (AWS and Google Cloud provisioning live; Azure provisioning in final validation — Azure pricing already live in the catalog)
Benchmark floors, location rules, placement affinity, spend caps	**Built**
Advisor (read-only savings counterfactual)	**Built** — install is a Helm chart
Extend your cluster (virtual node, burst tier)	**Early access** — design-partner phase
Burst partition for classic Slurm (ResumeProgram → pool)	Reference architecture — built with design partners on the mechanisms above
Hosted Slurm front door (Slinky + Open OnDemand + slurmrestd behind OIDC)	Reference architecture — standard upstream parts on the platform's serving stack, assembled per engagement
Slurm, Slinky, Open OnDemand, Cloudflare R2	Upstream projects and third-party services, used unmodified at arm's length (Slurm and Slinky stewarded by SchedMD — an NVIDIA subsidiary since Dec 2025, open-source continuation pledged)

Sources

Slurm: slurm.schedmd.com/power_save.html (elastic computing) · federation.html · rest.html (slurmrestd, JWT). Slinky: slurm.schedmd.com/slinky.html · github.com/SlinkyProject/slurm-operator · NVIDIA on Slinky (nvidia.com/en-us/software/slinky). Soperator: github.com/nebius/soperator. Open OnDemand: osc.github.io/ood-documentation (OIDC authentication). GROMACS scaling: Kutzner et al., *J. Comput. Chem.* 2025, doi:10.1002/jcc.70059. Cloudflare R2: developers.cloudflare.com/r2/pricing · r2/api/s3/api (S3 compatibility). Cloud egress: AWS/Azure/Google Cloud published bandwidth pricing (tiered, ~100 GB/month free; verify live calculators before quoting). Compute prices and CPU benchmark index: Multicloud live catalog, pulled 2026-07-22 — reproducible via the catalog API; benchmark scores are the catalog's per-CPU-model index, pooled across clouds. The field (section 11), per vendor documentation as of 2026-07-22: aws.amazon.com/pcs/pricing (rates cross-checked against the AWS Price List API) · AWS ParallelCluster · Azure CycleCloud Workspace for Slurm · Google Cluster Director (no service charge; underlying resources only) · CoreWeave SUNK · Nebius Soperator · Lambda Managed Slurm · Crusoe Managed Slurm · Rescale · SkyPilot · MemVerge · Exostellar. NVIDIA's acquisition of SchedMD: announced December 2025, nvidia.com/en-us/software/slurm and contemporaneous press. Platform: the Multicloud *Architecture Whitepaper* and docs/external/slurm-hpc-whitepaper.md (this paper's text source, with live links).

This paper describes a reference architecture on the platform as it stands, with status flags where a capability is early access or assembled per engagement. Prices are dated live-catalog pulls and public list prices — ask us to run the live catalog, or the Advisor read-only on your own environment, for current numbers on your workloads.

Contact: yaron@multicloud.io · multicloud.io