hivenet

The hybrid HPC playbook for scientists

When to run on cloud, when to stay on-prem, and how to cut costs without cutting results

Summary

Scientific work rarely fits a single box. Some models need low-latency interconnects and tight control. Others scale out and finish faster when you can spin up more machines. Many pipelines sit between those two.

This playbook helps you decide what runs where, how to control spend, and how to keep results reproducible. You get a decision framework, twelve proven patterns, cost guardrails that prevent surprises, and short runbooks you can apply this week.

On each page, you will find a fictional running example — Aurora Lab. A small hydrology group models regional floods. Their data lives in a national archive. Their local cluster's queue is a month long. A grant report is due in ten days. We'll follow their choices as we go in the orange boxes.

1. Who this is for and how to use it

This report is for people who make calls on compute without wanting a sales pitch. Principal investigators and lab managers will find budgeting guidance. Research software engineers will find patterns that translate into scripts and configs. R&D leads will get a shared language to align science, IT, and finance.

Short on time? Read the decision framework, workloads, patterns, and cost sections. Have an hour? Continue through data gravity, reproducibility, security, licensing, field notes, pilot plan, and the runbooks. Keep the tools and worksheets nearby. They turn choices into numbers.

Decide

- Who owns the first pilot.

Do

- Assign a reader for each section.
- Set a 30-minute review to pick one pattern.

Avoid

- Letting this become a long policy document.

Aurora's PI skims sections 3–6 to choose a path.
Their RSE bookmarks the runbooks for deployment.

2. The decision framework

This is the fork in the road. Answer five questions in order. Your workload tells you where it belongs.

1. **How tightly do tasks talk to each other?**
   Frequent MPI exchanges and latency-sensitive solvers want low-latency nodes on-prem or in an HPC-class environment. Independent tasks and sweeps are happy in the cloud or split across both.
2. **Where does the data live and how big is it?**
   If the dataset is large and stable, move code to the data. If inputs are small or short-lived, location is flexible.
3. **What is the time pressure?**
   Deadlines call for burst capacity. Flexible timelines can sit in a queue or use off-peak cloud windows.
4. **Any license, compliance, or export limits?**
   If a tool's license binds you to the campus network or country, keep those steps local or in a permitted region. If data is human or sensitive, plan for private or tightly isolated setups.
5. **What does your usage pattern look like over a year?**
   Spiky use fits pay-as-you-go. Constant heavy use fits owned or reserved capacity.

Outcome

Choose on-prem, cloud, or hybrid, plus a small first move you can defend.

Decide

- Who owns the first pilot.

Do

- Assign a reader for each section.
- Set a 30-minute review to pick one pattern.

Avoid

- Letting this become a long policy document.

Aurora marks the core solver as tightly coupled, the ensembles as independent, and the data as heavy. The framework points to a hybrid: solver local, ensembles in the cloud, post-processing next to the archive.

3. Workload archetypes

Names help. Once you name the shape of the job, the right pattern follows.

- **Tightly coupled MPI simulations.** Fluid dynamics, weather cores, and other solvers that pass messages often. These thrive on low latency and predictable nodes.
- **Embarrassingly parallel ensembles.** Parameter sweeps, Monte Carlo runs, bootstraps. Each task runs alone. Scale by running many at once.
- **GPU-heavy ML surrogates.** Training and inference for emulator models or pattern detection. These want short bursts of accelerators and a clean way to checkpoint.
- **Memory-bound analytics and graphs.** Genome assembly, large joins, big sparse graphs. These care about memory size and I/O patterns more than raw core counts.
- **Streaming or near-real-time.** Telescope feeds and sensor networks. These need steady ingest, small transforms at the edge, and aggregation centrally.
- **Mixed pipelines.** Simulation followed by analysis and visualization, often with different needs at each stage.

Decide

- The archetype for your next run.

Do

- Write it at the top of your job file.
- Share why it fits.

Avoid

- Treating all stages as the same job.

Aurora's pipeline is mixed: a solver that prefers low latency, then ensembles for scenarios, then post-processing of rasters for maps.

4. Twelve hybrid patterns that work in practice

Each pattern includes when to use it, what to avoid, and what to watch.

1. **Burst for deadlines**
   Spin up extra nodes for a fixed window. Checkpoint often. Shut everything down after.
   Use when: a hard date exists.
   Watch: idle spend and quota limits.
2. **Ensembles in cloud, calibration on-prem**
   Keep tight calibration loops on low-latency nodes. Fan out independent scenarios in the cloud.
   Use when: the core solver is chatty but scenarios are not.
   Watch: consistent seeds and environment parity.
3. **Code-to-data analysis**
   Run notebooks or jobs next to the big store. Export summaries and figures only.
   Use when: moving data is the bottleneck.
   Watch: access control, provenance, and small egress of final outputs.
4. **GPU on demand for surrogates**
   Train ML surrogates in short GPU bursts. Save checkpoints. Run inference near the data.
   Use when: surrogates replace slow inner loops.
   Watch: checkpoint storage and versioning.
5. **Memory-optimized nodes for omics or large graphs**
   Use fat-RAM machines for the spike in assembly or graph analytics. Turn them off after.
   Use when: jobs fail due to memory, not CPU.
   Watch: per-hour cost and spill to disk.
6. **Pre-stage shared references**
   Keep common datasets in one read-only location. Mount them for every job.
   Use when: many jobs read the same references.
   Watch: versioning and stable identifiers.
7. **Post-process in place, export small results**
   Keep raw outputs where they were produced. Push CSVs, thumbnails, and maps downstream.
   Use when: egress dominates cost and time.
   Watch: clear derivations and naming.

8. **Spot or preemptible runs with auto-checkpoint**
   Use discounted capacity for fault-tolerant tasks.
   Resume when interrupted.
   Use when: tasks can restart cleanly.
   Watch: checkpoint cadence and wall-time.
9. **License-aware split**
   Run licensed pre- or post-processors on-prem.
   Use open tools for the rest in the cloud.
   Use when: terms restrict where software runs.
   Watch: license server reach and geography clauses.
10. **Edge ingest to cloud aggregate**
    Filter and compress near sensors or instruments.
    Aggregate centrally for analysis.
    Use when: links are slow or expensive.
    Watch: time sync and schema drift.
11. **Reproduce-then-scale**
    Prove the pipeline on a tiny case locally. Run the same container at scale remotely.
    Use when: you need confidence before spending.
    Watch: identical images and pinned dependencies.
12. **Cloud sandbox for new hardware**
    Try new CPU or GPU types without buying them.
    Keep benchmarks simple and fair.
    Use when: hardware might change the model's economics.
    Watch: apples-to-apples comparisons.

Decide

- One pattern to try this month.

Do

- Write a half-page plan.
- Set a success metric and a spend cap.

Avoid

- Mixing three new patterns at once.

Aurora chooses 2 and 7. Solver and a licensed pre-processor stay local. Ensembles and post-processing run next to the archive. Only maps and small tables leave the region.

5. Cost and budget modeling

Cloud feels cheap until an idle cluster runs overnight. On-prem feels safe until you need ten times the cores for two weeks. Use simple math and a few guardrails.

Simple comparison

Annual cost ≈ core-hours + GPU-hours + storage + egress. If your cluster stays busy most of the year, owned or reserved capacity often wins. If your use is spiky, pay-as-you-go usually costs less.

Worked example A — burst vs buy (CPU-heavy month)

- Need: 500,000 core-hours in 3 weeks. Baseline is 40,000 core-hours per month.
- Cloud burst: 500,000 × €0.12 per core-hour = **€6,000** compute. Add storage €200 and egress €150. **Total €6,350.**
- Local: queue adds 6 to 8 weeks; buying extra nodes for a one-off is not viable.
- **Takeaway:** burst wins on time-to-result. Cost fits a grant buffer.

(Rates are illustrative.)

Worked example B — egress sanity check (data-heavy)

- Raw outputs: 12 TB. Post-processing reduces to 24 GB of CSV and PNG.
- Egress: 24 GB × €0.06 = **€1.44** versus 12 TB **≈ €720.**
- **Takeaway:** post-process next to data. Export summaries.

Outcome ranges you can expect

- Auto-stop and right-sizing: spend down **15 to 30%** in the first month.
- Ensemble bursting: time-to-result down **2 to 5×.**
- Code-to-data: egress spend down **90 to 99%** when exporting only derived artifacts.
- Reproduce-then-scale: failure on first large run down **50 to 70%.**

(Observed ranges from mixed lab deployments; results vary.)

Guardrails that pay for themselves

Tag every resource with project and owner. Set quotas and alerts at 50, 75, and 90 percent of budget. Turn on auto-stop for idle machines. After the first run, right-size instances.

Grant-friendly budgets

Split compute, storage, and egress on separate lines. Add a small buffer for re-runs and checkpoints. Note how the plan improves reproducibility and data sharing.

Decide

- Your spend cap for the pilot.

Do

- Turn on tags, alerts, and auto-stop now.
- Keep a one-page cost sheet per project.

Avoid

- “We will watch it manually.”

Aurora tags everything, sets auto-stop, and tracks egress weekly.
Their budget shows compute, storage tiers, and a small egress allowance.

6. Data gravity and egress

Moving data is the hidden cost. Uploading a terabyte can take days. Pulling it back can trigger fees that dwarf compute. Most labs do better when they stop moving data and start moving code.

Keep the dataset where it already lives. Launch jobs in the same region or facility. Use hot storage for active runs and colder tiers for archives. Lifecycle rules move files down the tiers when they age. Publish derived artifacts, not raw dumps.

Track egress monthly. A simple dashboard with a soft cap prevents late-month panic. If a team needs to pull more, they add a short note on why. That small step changes behavior.

Decide

- The home for your immovable dataset.

Do

- Run post-processing in that region.
- Export only derived artifacts.

Avoid

- Bulk downloads “for later.”

Aurora runs post-processing next to the archive.
They export maps and tables for the report. Raw rasters stay put.

7. Reproducibility and governance

Reproducible pipelines take stress out of collaboration.
The trick is to make sameness boring.

Put each stage in a container. Pin the operating system, libraries, and any CUDA stack. Describe steps and inputs in a short workflow file that lives with the code. Record provenance as you go: input hashes, random seeds, container digests, and a note on hardware. Map simple FAIR touchpoints. Give important datasets an identifier, a clear format, and access notes.

Decide

- What you will pin this week.

Do

- Containerize one stage.
- Add provenance logging.

Avoid

- “We will remember the versions.”

**Aurora keeps one image per stage. Workflow files live in git.
They record seeds and store input hashes next to outputs.**

8. Security and compliance basics

Research data deserves care even when it is public. Keep access narrow. Give people the minimum they need. Use private subnets. Avoid open ports unless you have a good reason. Encrypt at rest and in transit by default.

Do a light review before each project. Classify the data. Note any region rules or export limits. Sensitive or human data should live in private or dedicated setups with clear access records.

Compliance table (quick map)

Data type	Where to run	Controls	Notes
Public scientific	Cloud or on-prem	Private subnets, encryption	Prefer code-to-data for size
Human, de-identified	Private or isolated setups	IAM, audit logs, basic DLP scans	Review re-identification risk
Human, identifiable	On-prem or regulated private	Access reviews, key management, region pinning	Legal sign-off required
Export-controlled	On-prem or certified enclaves	Geo fencing, vetted users, logs	Document export paths

Decide

- The posture for your current dataset.

Do

- Write a one-page data handling note.
- Review access monthly.

Avoid

- Shared keys and public buckets.

Aurora's data is public. They still encrypt buckets, keep networks private, and audit access quarterly.

9. Licensing realities

Licenses set boundaries. Some tools bind to hostnames or networks. Some limit where software can run. Check the terms before you burst.

Licensing pre-flight checklist

- Host binding present
- License server reach and region
- Seat counts vs planned parallelism
- Cloud use permitted in terms
- Audit logs required
- Split plan for stages that must stay local

If the license allows cloud use, document the topology. If it does not, keep that stage on-prem and split the pipeline. Choose open tools for parts that scale out when you can.

Decide

- Which stages are license-bound.

Do

- Document the allowed topology.
- Plan the split for the next run.

Avoid

- Hoping the vendor will not mind.

Aurora's pre-processor stays on-prem due to license scope. The rest of the pipeline uses open tools that travel well.

10. Field notes by discipline

Short snapshots show how patterns land in real work.

Climate and weather

Core solvers stay on low-latency nodes. Ensembles and post-processing run next to shared archives. Turnaround drops from weeks to days. Egress stays small.

Genomics and bioinformatics

Assembly uses large-memory nodes for short bursts. References live in one read-only location. Queues shrink. Provenance improves because every job reads the same files.

Materials and computational chemistry

Teams try new accelerators in a sandbox. When a model responds, they scale screening runs as ensembles. More candidates get tested in the same time window.

Astrophysics and large surveys

Notebooks and jobs sit next to the data archive. People export catalogs and plots, not raw frames. Collaboration grows because no one needs to download terabytes.

Engineering CFD and FEA

Tight jobs run locally. Parameter sweeps burst on nights and weekends with checkpoints. Deadlines land without buying racks.

Decide

- The closest field note to your work.

Do

- Copy the pattern.
- Adjust only what is different.

Avoid

- Reinventing a working split.

Aurora borrows the climate pattern. They hit the report date and keep costs in line.

11. 30–60–90-day pilot blueprint

Start small, prove it, then scale. Do not bet the lab on day one.

Days 0–30. Prove a tiny case

- Containerize one stage; run a tiny case locally twice.
- Pre-stage references; turn on tags, alerts, and auto-stop.
- Add provenance logging.
- **Success:** identical outputs and spend under cap.

Days 31–60. Scale and burst

- Burst ensembles on nights or weekends with checkpoints.
- Move post-processing next to data; export summaries.
- **Success:** time-to-result meets target; egress under cap.

Days 61–90. Make it routine

- Add the licensing split; do a security review.
- Publish a one-page “Decide/Do/Avoid” for your lab.
- Schedule a quarterly spend and provenance audit.

Decide

- Your pilot owner and the day-90 targets.

Do

- Put the dates on a shared calendar.
- Share the success criteria.

Avoid

- Open-ended pilots.

12. Hybrid anti-patterns and fixes

Teams fall into the same traps. Skip them.

- **Lifting and shifting the whole cluster**
  You pay more for worse performance.
  Fix: start with ensembles or post-processing.
- **Unpinned environments**
  Results drift and cannot be reproduced.
  Fix: containers with pinned versions.
- **Open buckets and shared keys**
  Easy breach.
  Fix: least-privilege access, private subnets, short-lived creds.
- **No checkpointing on spot**
  Lost work on preemption.
  Fix: checkpoint at most every 20 minutes of compute.
- **Silent egress**
  Bills spike late month.
  Fix: monthly caps and a short review.

Decide

- One anti-pattern to remove this week.

Do

- Assign the fix.
- Verify in the next run.

Avoid

- “We will do it after the deadline.”

13. Quick-start runbooks

No theory here. Follow the steps. Adjust to your tools.

Burst without waste

Define target wall-time and cores. Containerize the app.
Tag the project and owner. Pre-stage inputs and references.
Launch nodes and run a small test. Enable checkpoints.
Monitor and right-size after the first pass. Post-process in place.
Export summaries. Shut everything down.

Set up code-to-data

Classify the dataset. Decide who can read and write.
Deploy a shared notebook or job runner next to the data.
Mount references read-only. Write outputs to a versioned bucket.
Capture provenance. Publish derived artifacts with a short README.

Put cost guardrails in place

Require tags. Turn on spend alerts. Auto-stop idle machines.
Review spend weekly. Close idle assets. Share a one-page policy so no one wonders what to do.

Decide

- Which runbook you start with.

Do

- Run a dry run today.
- Write what you changed.

Avoid

- Skipping the small test.

**Aurora follows the burst runbook. Ensembles finish on time.
Auto-stop prevents an overnight bill.**

14. Tools you can use this week

A few simple tools turn plans into action.

- **Readiness scorecard**
  Rate your data, licensing, security, and team skills. If a box is weak, fix it before you scale.
- **Workload fit score**
  Answer questions on coupling, data size, timeline, and cost tolerance. The score points to on-prem, cloud, or hybrid.
- **Calculators**
  A burst-versus-buy sheet and a small egress exposure model help you set expectations and budgets.
- **Checklists**
  Reproducibility, licensing, and security lists catch the basics. Print them and tape them up.

Decide

- Who owns each tool.

Do

- Fill them once.
- Review monthly.

Avoid

- Letting them sit in a folder.

15. Exit strategy to avoid lock-in

Design for the option to leave. Keep data in open formats. Keep workflows in portable languages. Keep everything inside containers. Avoid provider-specific pipelines at the core.
If you must use a proprietary service, write down an open fallback and the steps to move.

Decide

- Your open format for key outputs.

Do

- Write a short migration note.
- Store it with the code.

Avoid

- Assuming you will never move.

17. Takeaways

If you only do five things:

1. Name the job shape.
2. Put the data on a pedestal and move code to it.
3. Start with ensembles or post-processing.
4. Pin environments and log provenance.
5. Turn on auto-stop and egress caps.

What to measure next quarter: time-to-result, euro per valid run, percent of runs that reproduce on first try, egress per GB, idle hours.

Hivenet: a practical path for scientific computing

The themes in this report—scalability, reproducibility, cost control, and data stewardship—are exactly where Hivenet's distributed cloud takes a different approach. Instead of relying on centralized, energy-hungry data centers, Hivenet taps into underused devices across the globe. That shift changes the economics and the footprint of cloud for research groups.

- **Scalable compute for modeling:** Scientists can spin up GPU power (including high-performance cards like the RTX 4090 and 5090) for AI training, simulations, or parameter sweeps without waiting in cluster queues.
- **Data stays private and verifiable:** Files are encrypted, split into fragments, and distributed across the network. Only the researcher holds the keys, ensuring sensitive datasets remain under their control.
- **Costs that match research budgets:** With usage-based pricing and the option to contribute unused storage or compute in exchange for credits, labs can align spend with grants and avoid the unpredictable bills that plague traditional clouds.
- **Built for resilience and sustainability:** By eliminating the need for massive data centers, Hivenet reduces energy use by up to 77%. Workloads don't stall if one node goes offline; data is replicated across many contributors.

For labs like our fictional Aurora in our example, Hivenet offers a way to pilot hybrid models today: run ensembles and post-processing tasks on distributed GPUs, export only the summaries you need, and keep raw data safe at its source. It's a practical solution for labs that want the flexibility of cloud without compromising on privacy, reproducibility, or sustainability.