GPU SMITH
COMPUTE ENGINEERING

GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

AWS P5 Storage Cost: NVMe vs FSx for Lustre vs S3

> Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-09-19 · aws p5 storage cost · p5.48xlarge · local nvme · fsx for lustre · amazon s3 · gpu training storage · h100 · ml infrastructure · finops

Original article: https://gpusmith.com/articles/en/aws-p5-training-storage-cost

In this article

Executive Summary

For the defined scenario, **four p5.48xlarge nodes, 32 H100 GPUs, and a 40 TiB logical corpus per epoch**, the storage decision should be made on total paid job time, recovery behavior, and byte movement, not on storage price alone. Each p5.48xlarge has **eight 3.84 TB NVMe SSDs** (aws.amazon.com), and local instance store is included in the instance price (docs.aws.amazon.com). It is fast, node-local, and ephemeral. Data survives reboot but not stop, hibernation, termination, or replacement (docs.aws.amazon.com). A partitioned 40 TiB corpus fits comfortably across four nodes after a conservative reserve; four complete replicas do not.

As of **September 19, 2026**, AWS's dated N. Virginia example gives an On-Demand p5.48xlarge rate of **$55.04 per hour**, or **$220.16 per hour** for four nodes (aws.amazon.com). That makes compute waiting the decisive variable. At an illustrative, measured aggregate stage rate of 10 GiB/s, 40 TiB takes **1.138 hours**, making idle compute cost **$250.49** before productive training. This is a worksheet result, not a claimed benchmark. The live Capacity Blocks page listed **$41.528 per p5.48xlarge-hour**, but AWS says reservation prices update with supply and demand (aws.amazon.com). Buyers must snapshot the actual quote.

FSx for Lustre is the strongest shared POSIX option when the workload needs one namespace, coordinated shuffles, and predictable restart without rebuilding every node cache. Scratch storage is temporary and not replicated (docs.aws.amazon.com), while persistent deployments add replication and automated component replacement. AWS's N. Virginia examples price scratch SSD at **$0.140 per GB-month** and Persistent 2 at **$0.145 per GB-month** (aws.amazon.com) (aws.amazon.com). The 40 TiB / 40,960 GiB figures are abstract per-GiB prorating examples, not deployable FSx configurations. Persistent SSD creation supports 1,200 GiB, 2,400 GiB, and then 2,400-GiB increments, while an EFA-enabled 125 MBps/TiB configuration must use 38,400-GiB increments. Select whether EFA is enabled and price a valid capacity, including throughput and metadata, before treating it as a job-cost estimate.

S3 Standard should remain the authoritative durable copy. Its official example rate is **$0.023 per GB-month**, and GET requests are **$0.0004 per 1,000** (aws.amazon.com) (aws.amazon.com). Direct S3 streaming can be economical when shards are large, access is sequential, cache hits are high, and telemetry proves the GPUs are not waiting. It is not automatically a POSIX parallel file system: Mountpoint intentionally omits full POSIX

semantics (docs.aws.amazon.com). The default recommendation is a measured hybrid: S3 as system of record, local NVMe for hot shards and temporary checkpoint assembly, and FSx only where a shared namespace or recovery objective justifies it.

Introduction and Background

An AWS P5 storage comparison is often reduced to three price labels: bundled NVMe, FSx dollars per GB-month, and S3 dollars per GB-month. That framing misses the most expensive meter. AWS bills On-Demand EC2 while instances are running, with per-second billing after a **60-second minimum** (docs.aws.amazon.com). If staging, small-file metadata, cache misses, or checkpoint flushes leave four P5 nodes idle, the compute line can dominate the storage line.

This report freezes one explicit engineering scenario rather than presenting a market benchmark. The cluster has **four p5.48xlarge instances**, each with eight H100 GPUs, for **32 GPUs** total. The training loop scans **40 TiB of logical data per epoch**. The corpus is sharded, but object count and size distribution must be recorded. The job creates distributed checkpoints, has a buyer-defined GPU-idle ceiling, and must meet a stated recovery-time objective. AWS lists up to **3,200 Gbps of Elastic Fabric Adapter networking** for P5 (aws.amazon.com), but a network maximum is not an application data rate.

GPU Smith is an adjacent independent engineering adviser, not a cloud or storage provider. Its published method says infrastructure is validated against written acceptance criteria (gpusmith.com) and describes workload modeling and total-cost comparison as assessment deliverables (gpusmith.com). That posture fits this decision: specify the workload, measure each path with the same seed and shard layout, and accept or reject an architecture against cost, utilization, integrity, and recovery thresholds.

Workload Definition and Complete Byte Map

The design record should freeze more than capacity. It must state Region, Availability Zone, AMI, filesystem, RAID layout, shard count, median and tail object size, shuffle behavior, epoch count, checkpoint bytes and cadence, restart point, and the maximum acceptable recovery time. PyTorch defines `prefetch_factor` as the number of batches loaded in advance per worker (docs.pytorch.org), so worker count and prefetch settings are cost-model inputs, not incidental tuning.

Every physical copy should then be mapped from creation through deletion. *Table 1* shows the minimum byte-flow ledger for this scenario.

Flow	Logical bytes and multiplier	Frequency	Charge category	Durability and completion evidence
S3 authoritative corpus to four local caches	40 TiB total if shards are partitioned; 160 TiB if every node receives a full copy	Initial stage and each destructive replacement	P5 waiting time, S3 GETs, endpoint path	Local copy is disposable; record checksum and completed-shard manifest
S3 to FSx	Up to 40 TiB materialized, possibly lazy	Initial load or first access	FSx storage, requests, P5 wait, possible S3 requests	S3 remains authoritative; verify load state before timing

Flow	Logical bytes and multiplier	Frequency	Charge category	Durability and completion evidence
S3 direct to workers	40 TiB per epoch before cache hits	Every epoch	GETs, endpoint processing, GPU data-wait	S3 durable; local cache disposable
Training ranks to local checkpoint assembly	`checkpoint_bytes` across ranks	Each checkpoint	P5 compute overlap or pause	Unsafe as sole copy after stop or replacement
Checkpoint to FSx or S3	`checkpoint_bytes` plus retries	Each committed checkpoint	FSx writes or S3 PUTs, possible network path	Commit only after all rank shards and metadata are durable
FSx export to S3	Changed files and checkpoint bytes	At milestones and teardown	Export requests, P5 or FSx time	Wait for the export queue to drain before deletion

The important distinction is between **logical bytes** and **physical bytes moved**. Partitioned local staging moves 40 TiB once. Full replication moves 160 TiB and does not fit on each node's nominal 30.72 TB. Lazy FSx import may make metadata visible before data blocks are resident: AWS says file content loads on first application access (docs.aws.amazon.com). Teardown is also a byte path. Automatic export writes S3 Standard objects, and AWS advises waiting for `AgeOfOldestQueuedMessage` to reach zero before deletion (docs.aws.amazon.com).

Local NVMe Staging

Capabilities

The **p5.48xlarge** exposes eight nominal **3.84 TB NVMe SSDs**, or 30.72 TB decimal per node. AWS specifies its published instance-store random IOPS at **4,096-byte blocks and queue-depth saturation** (docs.aws.amazon.com). That is a device test condition, not a promise that a sharded training pipeline will deliver its batches at that rate.

The capacity-fit equation is:

```
required_local_bytes_per_node = assigned_shards + cache_headroom + active_checkpoint_bytes + filesystem_reserve
```

With even partitioning, assigned shards are 10 TiB per node. AWS recommends leaving **10%** of SSD instance-store capacity unpartitioned to reduce write amplification (docs.aws.amazon.com). An ext-family filesystem also defaults to a **5% block reserve** unless configured otherwise (man7.org). Those policies overlap differently by layout, so the actual mounted usable bytes must be measured after formatting. Teams should not subtract percentages from a marketing capacity and call that a verified result.

Strengths and Limitations

Local NVMe has no separate storage rental line while the instance runs. It also avoids a shared remote filesystem in the steady-state hot path. Parallel staging can start all four nodes together, and a node can read its assigned shards without cross-node storage traffic. RAID 0 can aggregate devices because I/O is striped

across members (docs.aws.amazon.com), but one lost member loses the complete array. For a disposable cache, that can be acceptable if automation rebuilds it and checkpoints live elsewhere.

The durability boundary is strict:

- **Reboot:** instance-store data normally persists.
- **Stop, hibernate, terminate:** data does not persist.
- **Automatic recovery or node replacement:** the cache must be rebuilt.
- **Disk failure:** data on the failed disk is lost.
- **Encryption:** AWS encrypts local NVMe using XTS-AES-256 in hardware (docs.aws.amazon.com). Customer-managed keys are not available for this instance-store layer.

Local staging is therefore attractive when the stage completes quickly relative to the job, a replacement can re-stage inside the recovery objective, and the data-loader benefits materially from a warm cache. It is weak when every node needs the entire corpus, checkpoints cannot be exported promptly, or a single replacement would miss the service-level target.

FSx for Lustre

Capabilities

Amazon FSx for Lustre provides a shared POSIX namespace and two deployment families, persistent and scratch (docs.aws.amazon.com). SSD, HDD, and Intelligent-Tiering are documented storage classes. Persistent SSD and HDD replicate within one Availability Zone, while Intelligent-Tiering replicates across Availability Zones (docs.aws.amazon.com).

Persistent 2 SSD exposes **125, 250, 500, and 1,000 MBps per TiB** throughput tiers (docs.aws.amazon.com). Intelligent-Tiering throughput is provisioned in **4,000 MBps increments**. EFA-enabled FSx supports GPUDirect Storage, and P5 is a supported client (docs.aws.amazon.com). This should still be treated as a compatibility capability. AWS explicitly says achieved throughput and IOPS depend on workload characteristics in addition to provisioned resources (docs.aws.amazon.com).

Strengths and Limitations

FSx simplifies global shuffling and checkpoint visibility because every rank can address one namespace. It can also link to S3, materialize data lazily, preload files in parallel, and export changed content. This reduces custom staging logic, but it does not remove the need to distinguish cold and warm reads. First access to nonresident S3-linked content adds latency (docs.aws.amazon.com).

The cost model must include more than capacity:

- **Storage:** provisioned SSD GB-month or consumed Intelligent-Tiering GB-month.
- **Throughput:** per-unit storage throughput or provisioned Intelligent-Tiering MBps.
- **Metadata:** included and additional metadata IOPS, sensitive to file count and directory shape.

- **Requests and cache:** Intelligent-Tiering reads, writes, and optional SSD cache.
- **Protection:** persistent backup storage where supported.
- **Transfer:** traffic outside the preferred Availability Zone.
- **Lifecycle:** time from create through verified export and deletion.

Availability Zone alignment is financially material. Access in the preferred AZ has no FSx transfer charge, while same-Region access from another AZ is **$0.01 per GB in each direction** (aws.amazon.com). One 40 TiB read path outside the preferred AZ is therefore about **$409.60** before return traffic. EFA clients and FSx also need the same Availability Zone and `/16` CIDR (docs.aws.amazon.com).

Scratch is appropriate when S3 is authoritative and a file-server loss can be recovered by rebuilding. Persistent is preferable when the file system itself must survive component replacement and restoration must be faster. AWS says backups are unavailable for scratch and for S3-linked file systems (docs.aws.amazon.com), so "linked to S3" and "backed up" must not be assumed equivalent.

S3 Streaming and Local Cache

Capabilities

Amazon S3 Standard is the durable object-store baseline. AWS designs it for **99.999999999% durability** and **99.99% availability** and stores data across at least three Availability Zones (docs.aws.amazon.com). Same-Region transfer from S3 to EC2 has no S3 data-transfer charge (aws.amazon.com). A gateway VPC endpoint has no additional endpoint charge, while an interface endpoint and NAT gateway can add data-processing charges.

S3 offers strong read-after-write consistency for PUT and DELETE requests (docs.aws.amazon.com), multipart upload for large objects, and Range GET for partial reads. AWS suggests considering multipart upload at **100 MB** (docs.aws.amazon.com). It also states that actual performance varies with workload and configuration, which is why request-rate guidance cannot be converted into training throughput.

Strengths and Limitations

Direct streaming removes a full up-front copy and can overlap fetch with compute. It performs best when records are packed into substantial sequential shards, workers prefetch predictably, and the same data is reused from a local cache in later epochs. AWS notes that a single GET can retrieve multiple samples from a large shard (aws.amazon.com). PyTorch warns that inefficient loading can leave expensive hardware idle (docs.pytorch.org).

Mountpoint for S3 translates file operations into object API calls (docs.aws.amazon.com). It can use local EC2 instance storage as a cache and is optimized for sequential reads of large objects (github.com). It cannot modify existing files or delete directories. A training pipeline must validate that its open, seek, rename, append, and checkpoint patterns fit those semantics.

The request-cost example is revealing. Forty TiB contains **41,943,040 one-MiB objects**, producing about **$16.78** of S3 Standard GET charges per full scan at $0.0004 per 1,000 GETs. The same bytes packed into **40,960 one-GiB shards** cost about **$0.016** in GET requests. Request dollars are small beside P5 waiting cost, but object count also affects latency, CPU work, connection pressure, and metadata operations.

Network path selection matters more. AWS's examples price interface endpoint processing at **$0.01 per GB** and NAT gateway processing at **$0.045 per GB** (aws.amazon.com) (aws.amazon.com). Applied to 40 TiB, those illustrative paths add about **$409.60** and **$1,843.20** per scan. A gateway endpoint avoids that endpoint processing charge (docs.aws.amazon.com).

Feature Comparison

Table 2 compares the three contenders on the same decision axes. S3 is included as object storage, not mislabeled as a parallel POSIX filesystem.

Criterion	Local NVMe stage/cache	FSx for Lustre	S3 stream/cache
Separate storage price	Bundled with P5 while instance runs	Capacity plus applicable throughput, metadata, requests, cache, backup, and transfer	Storage plus requests, retrieval where applicable, and endpoint path
Namespace	Node-local; application must partition or replicate	Shared POSIX namespace	Object namespace; Mountpoint has partial POSIX semantics
Cold-start path	Full or selective copy to every required node	Preload or first-access materialization	Immediate streaming, optionally filling local cache
Steady-state strength	Very low-latency local reads after warm-up	Coordinated shared access and distributed checkpoint visibility	Durable system of record and elastic object access
Failure boundary	Lost on stop, termination, replacement, or local disk failure	Scratch can lose data on server failure; persistent adds replication	Multi-AZ durable object copy
Recovery action	Replace node, rebuild filesystem, re-stage shards, load durable checkpoint	Remount or restore according to deployment; validate export/backup	Recreate clients/cache and resume from committed object checkpoint
Best fit	Long jobs with reusable hot data and acceptable re-stage time	Shared POSIX, global shuffle, high checkpoint coordination, bounded restart	Large sequential shards, high cache hit rate, durable source and sink
Primary cleanup risk	Checkpoints left only on ephemeral media	File system left running or export incomplete	Orphan multipart uploads, excess versions, or cache lifecycle drift

No option wins without measured data. NVMe can minimize steady-state latency yet lose on repeated re-stage. FSx can cost more as a storage line but reduce expensive GPU waiting and operator effort. S3 can have negligible request cost but still be the most expensive path if first-byte latency and small objects repeatedly starve workers.

Performance and Benchmarks

The pilot must use the **same corpus layout, model seed, sample order, worker count, checkpoint size, and Region/AZ topology** for every contender. It should run cold-cache and warm-cache phases, record actual elapsed wall time, and separate staging, productive compute, input wait, checkpoint pause, and recovery. GNU time can record elapsed real wall-clock time around a stage or checkpoint command (gnu.org).

Synthetic tests are diagnostics, not the purchasing result:

- **fio:** verify achieved queue depth rather than assuming the requested value (fio.readthedocs.io). Capture bandwidth, IOPS, latency distributions, and device busy time.
- **IOR:** use distributed read and write patterns, but prevent same-node readback from measuring cache instead of the filesystem (github.com).
- **mdtest:** characterize create, stat, and delete pressure for the real directory and file-count distribution. The Lustre project identifies mdtest as an MPI metadata benchmark (wiki.lustre.org).
- **Application trace:** label data fetch, host-to-device copy, forward/backward, optimizer, and checkpoint phases with PyTorch Profiler, which supports user-named code ranges (docs.pytorch.org).
- **GPU telemetry:** correlate application throughput with device-memory activity and PCIe or NVLink traffic. NVIDIA says DCGM profiling values are interval averages, not kernel traces (docs.nvidia.com).
- **Cache telemetry:** count lookups, hits, misses, evictions, bytes filled, and latency. Prometheus recommends cache query volume, hits, and overall latency as key metrics (prometheus.io).

Histograms should preserve latency distributions for stage operations, batch waits, and checkpoint commits. OpenTelemetry histograms record count, sum, and bucket observations (opentelemetry.io). Prometheus counters should be converted to per-second rates, not compared as raw cumulative values (prometheus.io).

Cache state must be explicit. fio can invalidate page-cache entries before a controlled cold run (fio.readthedocs.io), while the Linux kernel warns that dropping caches can itself add substantial I/O and CPU cost and is intended for testing (kernel.org). Record the exact procedure rather than claiming a run was simply "cold."

Reproducible Pilot Protocol

The pilot should produce a machine-readable evidence bundle, not a screenshot of one average. For each run, record the configuration digest, workload seed, shard manifest, cache state, start and completion timestamps, process exit status, P5 instance IDs, FSx ID if used, and the S3 request namespace. Instrument batch waits as histograms. OpenTelemetry histograms retain population count (opentelemetry.io), the sum of recorded values (opentelemetry.io), and observation counts per bucket. Boundaries should be selected so worst-case measurement error is within the team's tolerance (opentelemetry.io). This makes a p95 batch-wait target auditable rather than decorative.

Metric types must remain semantically correct. A cumulative counter becomes useful after a rate calculation (prometheus.io), but Prometheus warns not to apply `rate()` to a gauge (prometheus.io). For last-success events, export the event's Unix timestamp (prometheus.io). Prometheus defines the 0.95 quantile as the 95th

percentile (prometheus.io), and its histogram guidance says bucket boundaries should fit both the expected range and intended queries (prometheus.io). Calculate quantiles from histogram buckets at query time (prometheus.io).

Device diagnostics need comparable settings. fio defines I/O depth as the number of I/O units kept in flight (fio.readthedocs.io). It also notes that increasing depth above one does not change synchronous engines (fio.readthedocs.io). Preserve full latency buckets with JSON+ output rather than an average alone (fio.readthedocs.io). GNU time can independently capture filesystem input and output operation counts (gnu.org) (gnu.org). Its wall-clock measure should wrap the same staging boundary for every contender (gnu.org).

Cold-cache tests require care. Linux can drop clean caches and reclaimable slab objects (kernel.org), but does not discard dirty objects (kernel.org). The kernel documentation warns that the operation can create performance problems (kernel.org). Run it only in the isolated pilot, synchronize dirty data first, and record its completion before starting the timed interval.

Operating-system counters are cross-checks, not the application verdict. Linux exposes device counters through `/proc/diskstats` and `/sys/block` (docs.kernel.org). Almost all fields are cumulative, so before-and-after snapshots need reset handling (docs.kernel.org). The kernel also supplies a block-I/O completion tracepoint (docs.kernel.org). `iostat` defines await as elapsed milliseconds for requests (man7.org), including queue and device service. It cautions that utilization alone does not reveal the performance limit of RAID arrays and modern SSDs (man7.org).

Capacity evidence should be captured from the mounted system after creation. The ext-family formatter bases filesystem size on device size when no explicit size is given (man7.org). Larger inodes consume more inode-table space (man7.org). Reserve settings, RAID metadata, inode policy, and cache headroom therefore belong beside the measured free-byte result.

For FSx, benchmark the stack in layers. The Lustre project says performance depends on hardware, networking, and the operating system (wiki.lustre.org). Its process builds a view from network, device, server, metadata, and application measurements (wiki.lustre.org). Compare results with a recorded reference, as the project recommends (wiki.lustre.org). Size metadata from planned file count (lustre.org) and object storage from planned data volume (lustre.org).

IOR provides application-level bandwidth evidence and mdtest provides metadata evidence. IOR defines bandwidth from transferred bytes divided by elapsed time (github.com). Its companion documentation says mdtest measures peak metadata rates (ior.readthedocs.io). Run both with the same clients, stripe policy, file count, and access pattern as the application, then retain the raw command line and output.

Finally, correlate storage events with GPU activity. NVIDIA DCGM can expose device-memory activity and PCIe or NVLink traffic (docs.nvidia.com). NVIDIA advises comparing related metrics over a representative interval and correlating them with throughput (docs.nvidia.com). Nsight Systems can collect storage throughput and operation metrics (docs.nvidia.com), place labeled CPU regions on its timeline (docs.nvidia.com), and project them onto the GPU timeline (docs.nvidia.com). A storage winner is the option that passes the GPU-idle ceiling in this application trace, not the option with the largest synthetic headline.

The evidence bundle should end with an audit checklist. Confirm that histogram sums reconcile with event counts (opentelemetry.io), command-level filesystem outputs were retained (gnu.org), and IOR is treated as a filesystem benchmark rather than an application result (wiki.lustre.org). Confirm that cache dropping stayed inside the test environment (kernel.org), disk-counter snapshots came from documented kernel interfaces (docs.kernel.org), and cumulative counters were differenced correctly (docs.kernel.org).

Also retain `await`, which represents elapsed request time (man7.org), the formatted reserve policy (man7.org), and the inode-size decision (man7.org). Correlate memory and fabric activity (docs.nvidia.com) over a representative interval (docs.nvidia.com) with storage operations (docs.nvidia.com). Finally, show that metadata capacity follows file count (lustre.org), data capacity follows total bytes (lustre.org), and mdtest was used only for metadata rates (ior.readthedocs.io).

The recovery record must be equally concrete. Save enough data-loader state to skip already processed batches after a restart (huggingface.co). Confirm that a recreated Pod receives a newly created ephemeral volume (kubernetes.io), and verify that local-storage limits cannot trigger an unexpected eviction (kubernetes.io). These checks turn the restart requirement into evidence.

Data Analysis and Evidence

As of the publication date, the official dated On-Demand input is **$55.04 per p5.48xlarge-hour**, making the four-node aggregate **$220.16 per hour**. The live Capacity Blocks table lists **$41.528 per instance-hour**, or **$166.112 per cluster-hour**, but Capacity Blocks are scheduled and charged up front (aws.amazon.com). These purchase models are not interchangeable. The buyer should insert the committed rate actually available for the planned window.

The core formulas are:

- **Stage-time lower bound:** `bytes_to_stage / measured_sustained_stage_rate`.
- **GPU wait cost:** `aggregate_instance_hourly_rate * measured_hours_staging_or_data_stalled`.
- **FSx job cost:** prorated storage plus throughput, metadata, requests, cache, backup, transfer, and delayed teardown.
- **S3 path cost:** retained storage plus requests, retrieval, endpoint processing, and measured compute wait.
- **Full job TCO:** productive compute plus waiting compute plus storage plus requests plus transfer plus expected recovery cost.

Table 3 is a sensitivity worksheet, not a performance claim. It holds bytes at 40 TiB and varies only the **measured aggregate** stage rate. Cost uses the dated On-Demand four-node rate.

Measured aggregate stage rate	Elapsed stage time	Four-node waiting cost	Interpretation
5 GiB/s	2.276 hours	$500.99	A storage optimization worth $100 can be economical if it removes only 27.3 minutes of cluster wait

Measured aggregate stage rate	Elapsed stage time	Four-node waiting cost	Interpretation
10 GiB/s	1.138 hours	$250.49	Baseline illustration; must be replaced with pilot telemetry
20 GiB/s	0.569 hours	$125.25	Doubling rate saves about $125.24 on this one stage
40 GiB/s	0.284 hours	$62.62	Further gains matter less in dollars because the stage window is already short

The table demonstrates why vendor maxima are unsuitable inputs. Even a correct maximum says nothing about shard sizes, decompression, worker scheduling, checksums, page cache, connection count, or data-loader CPU time. Linux `iostat` defines `await` to include queue and service time (man7.org), while Linux exports cumulative per-device counters through procfs and sysfs (docs.kernel.org). Those are useful cross-checks, but application timestamps determine the paid stall.

Do not use the 40,960-GiB FSx worksheet as deployable configuration pricing. Choose the FSx deployment and whether EFA is enabled, then use the permitted capacity increment and recompute FSx storage, throughput, metadata, requests, backup, transfer, and lifecycle costs. The S3 allocation remains a separate illustrative retention calculation.

The FSx values exclude all additional dimensions. AWS prices extra metadata IOPS at **$0.055 per IOPS-month** in its example (aws.amazon.com). FSx usage is prorated by the second and has no setup charge (aws.amazon.com), so disciplined teardown is a real control.

The sensitivity grid should also cross variables that dollars per GB omit:

- **Epoch count:** more epochs favor a reusable local or FSx cache.
- **Cache hit rate:** lower hits move more S3 bytes and increase batch-wait variance.
- **File size and count:** smaller objects increase request count and metadata pressure.
- **Checkpoint cadence:** frequent checkpoints increase writes but reduce lost work after restart.
- **Node replacements:** each replacement adds re-stage bytes and time for local caching.
- **Delayed teardown:** every extra hour adds FSx and possibly compute cost.
- **AZ mismatch:** each directed 40 TiB FSx transfer adds about $409.60 at the cited rate.

Recovery should be tested, not inferred. PyTorch Distributed Checkpoint saves and loads from multiple ranks concurrently and can reshard across a changed topology (docs.pytorch.org) (docs.pytorch.org). A complete recovery test must also validate optimizer, random-number generator, gradient scaler, data position, and sample order. Hugging Face Accelerate explicitly lists model, optimizer, random-number generators, and gradient scaler in checkpoint state (huggingface.co).

Implications and Future Directions

The most defensible default is a **measured hybrid**. Keep the authoritative corpus and committed checkpoints in S3. Use local NVMe for assigned hot shards, decoded samples, and temporary checkpoint assembly. Add FSx when the application truly needs shared POSIX behavior, when global reshuffling makes partitioned caches awkward, or when its recovery-time improvement is worth more than its complete incremental cost.

The procurement decision record should require:

- **A frozen scenario:** Region, AZ, exact instance type, image, filesystem, network path, object distribution, epochs, and checkpoint policy.
- **A byte-flow ledger:** every source, destination, multiplier, frequency, charge, and delete/export action.
- **Measured acceptance criteria:** p50, p95, and p99 batch-wait latency, stage wall time, effective GiB/s, GPU idle percentage, checkpoint commit time, and restart time.
- **Integrity checks:** corpus manifests, S3 checksums where used, sample-order validation, and a restored-step equivalence test.
- **Cost evidence:** CUR or billing-export categories reconciled with application request counts and elapsed resource time.
- **Failure drills:** one node replacement, one cold restart, one cache-miss surge, and one delayed checkpoint/export case.

Kubernetes users should treat node-local cache as ephemeral. Kubernetes documents that ephemeral volumes follow the Pod lifetime (kubernetes.io) and may evict a Pod that exceeds local ephemeral-storage limits (kubernetes.io). Capacity requests, cleanup, and restart automation belong in the storage design.

Checkpoint overlap also needs explicit control. PyTorch says users manage concurrently running asynchronous checkpoints (docs.pytorch.org). A second save should not silently race a prior export or overwrite the only recoverable generation. Maintain immutable generations, publish a completion marker only after all shards pass validation, retain at least the prior committed generation, and measure the real pause or overlap cost.

Future price changes do not invalidate this method. Storage, compute, endpoint, and transfer rates should be versioned inputs. What remains stable is the accounting structure: map the bytes, measure the time, price the complete path, and test the recovery promise.

Conclusion

For this four-node P5 scenario, **local NVMe is the best candidate for a hot cache**, not a durable system of record. A partitioned 40 TiB corpus fits across the cluster with substantial headroom, while full per-node replication does not. Its apparent storage cost is zero only in the narrow sense that the devices are bundled. Initial staging, destructive replacement, checkpoint export, and GPU waiting remain billable through P5 runtime.

FSx for Lustre is the strongest shared-filesystem option when POSIX semantics, coordinated access, and recovery time justify the extra dimensions of capacity, throughput, metadata, requests, backup, transfer, and lifecycle. Scratch should be modeled as rebuildable. Persistent should be selected when its replication and recovery behavior meet a requirement that S3 plus disposable cache cannot meet as simply.

S3 should remain the durable source and checkpoint destination. Direct streaming is viable when shard size, concurrency, prefetch, and cache behavior keep measured GPU data-wait below the buyer's ceiling. A gateway endpoint and large sequential shards can keep request and path charges small, but only application telemetry can establish whether the result feeds 32 H100 GPUs efficiently.

The decision is therefore not NVMe versus FSx versus S3 in isolation. It is a choice among complete staging and recovery cycles. The winning design is the least-cost path that passes the same workload, integrity, utilization, and recovery acceptance tests. For many teams, that will be S3 durability, partitioned NVMe caching, and selective FSx use where a shared namespace has measurable value.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://aws.amazon.com/ec2/instance-types/accelerated-computing/
2. https://docs.aws.amazon.com/dlami/latest/devguide/aws-deep-learning-base-gpu-ami-ubuntu-20.04.html
3. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-store-lifetime.html
4. https://gpusmith.com/articles/en/aws-gpu-instance-pricing
5. https://aws.amazon.com/blogs/machine-learning/secure-short-term-gpu-capacity-for-ml-workloads-with-ec2-capacity-blocks-for-ml-and-sagemaker-training-plans/
6. https://aws.amazon.com/ec2/capacityblocks/pricing/
7. https://docs.aws.amazon.com/fsx/latest/LustreGuide/using-fsx-lustre.html
8. https://aws.amazon.com/fsx/lustre/pricing/
9. https://aws.amazon.com/blogs/database/automate-postgresql-audit-log-extraction-and-analysis-with-amazon-s3/
10. https://aws.amazon.com/s3/pricing/
11. https://docs.aws.amazon.com/AmazonS3/latest/userguide/mountpoint-usage.html
12. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-on-demand-instances.html
13. https://aws.amazon.com/ec2/instance-types/p5/
14. https://gpusmith.com/articles/en/gpu-cluster-acceptance-testing-runbook
15. https://gpusmith.com/
16. https://docs.pytorch.org/docs/main/data.html
17. https://docs.aws.amazon.com/fsx/latest/LustreGuide/importing-files-dra.html
18. https://docs.aws.amazon.com/fsx/latest/LustreGuide/autoexport-data-repo-dra.html
19. https://docs.aws.amazon.com/ec2/latest/instancetypes/ac.html
20. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ssd-instance-store.html
21. https://man7.org/linux/man-pages/man8/mke2fs.8.html

22. https://docs.aws.amazon.com/ebs/latest/userguide/raid-config.html
23. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/data-protection.html
24. https://docs.aws.amazon.com/fsx/latest/LustreGuide/managing-throughput-capacity.html
25. https://docs.aws.amazon.com/fsx/latest/LustreGuide/efa-file-systems.html
26. https://docs.aws.amazon.com/fsx/latest/LustreGuide/preload-file-contents-hsm-dra.html
27. https://docs.aws.amazon.com/fsx/latest/LustreGuide/using-backups-fsx.html
28. https://docs.aws.amazon.com/AmazonS3/latest/userguide/DataDurability.html
29. https://aws.amazon.com/s3/faqs/
30. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html?sc_channel=el&trk=1ad04439-1c50-4fdd-a845-d07d2655fe7a
31. https://docs.aws.amazon.com/AmazonS3/latest/userguide/qfacts.html?trk=article-ssr-frontend-pulse_little-text-block
32. https://aws.amazon.com/blogs/machine-learning/applying-data-loading-best-practices-for-ml-training-with-amazon-s3-clients/
33. https://docs.pytorch.org/tutorials/intermediate/intermediate_data_loading_tutorial.html
34. https://docs.aws.amazon.com/AmazonS3/latest/userguide/mountpoint.html
35. https://github.com/awslabs/mountpoint-s3/blob/main/doc/SEMANTICS.md
36. https://aws.amazon.com/privatelink/pricing/
37. https://aws.amazon.com/vpc/pricing/?did=ap_card
38. https://docs.aws.amazon.com/vpc/latest/privatelink/vpc-endpoints-s3.html
39. https://www.gnu.org/software/time/manual/time.html
40. https://fio.readthedocs.io/en/latest/fio_doc.html
41. https://github.com/hpc/ior/blob/main/doc/USER_GUIDE
42. https://wiki.lustre.org/Category:Benchmarking
43. https://docs.pytorch.org/tutorials/recipes/recipes/profiler_recipe.html
44. https://docs.nvidia.com/datacenter/dcgm/latest/learn/modules/profiling.html
45. https://prometheus.io/docs/practices/instrumentation/
46. https://opentelemetry.io/docs/specs/otel/metrics/data-model/
47. https://kernel.org/doc/html/v5.18/admin-guide/sysctl/vm.html
48. https://prometheus.io/docs/practices/histograms/
49. https://docs.kernel.org/admin-guide/iostats.html
50. https://docs.kernel.org/core-api/tracepoint.html
51. https://man7.org/linux/man-pages/man1/iostat.1.html
52. https://www.lustre.org/getting-started-with-lustre/
53. https://ior.readthedocs.io/en/latest/
54. https://docs.nvidia.com/nsight-systems/UserGuide/
55. https://huggingface.co/docs/accelerate/v1.7.0/en/usage_guides/checkpoint
56. https://kubernetes.io/docs/concepts/storage/ephemeral-volumes/
57. https://kubernetes.io/docs/concepts/storage/ephemeral-storage/

58. https://docs.pytorch.org/tutorials/recipes/distributed_checkpoint_recipe.html

59. https://docs.pytorch.org/tutorials/recipes/distributed_async_checkpoint_recipe.html

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the hardware library, GPU references, networking references and vendor research. Contact GPU Smith with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: https://gpusmith.com