PURETENSOR
INTELLIGENCE DISTILLED

The Zombie Mount

Lazy unmount hid a dead network mount twice, and only one of the two failures is explained

Paper: PT-R-2026-062 v1.0

Author: PureTensor Ltd

Date: 2 November 2026

Status: Published

Abstract

A lazy unmount (`umount -l`) removes a mounted filesystem's name from the mount namespace, returns success, and leaves the kernel object alive. Every tool that asks whether the path is mounted then answers no, including the monitoring that was supposed to notice a dead filesystem. We found two incidents on one host in 2026 in which a network mount that was dead (June) or evicted (November) had been lazily detached before the loss of the whole node. Only the second is shown to depend on the detachment.

On 23 June the node, a 503 GiB, four-GPU machine with no swap at the time, froze silently at 12:34 UTC: no panic, no out-of-memory kill, and a journal that ended cleanly. According to the commit message of the fix, our reconciler script had been lazily unmounting a dead NFS-over-RDMA mount for about five hours. Our first diagnosis was that pinned dirty pages from that mount livelocked memory reclaim. The long-term Prometheus record, which we reconstructed only for this paper, does not support that. Dirty memory stayed at or below 0.4 GiB, writeback was at most 4,096 bytes, unstable NFS pages were zero, and private process memory rose from 45.9 GiB to 392.5 GiB in six minutes in processes outside the Kubernetes pods. The zombie mount is a plausible co-factor on 23 June, not a demonstrated cause, and the allocator was never identified.

On 2 November an evicted kernel CephFS client was recovered with a forced and then a lazy unmount. The old superblock survived in private mount namespaces; 98 seconds later a process exit tore it down with busy inodes, and kernel 7.0.0-38-generic panicked. Our notes record this as a use-after-free, inferred second-hand from a poison address. Available memory showed no shortage in the hour before, although 68 to 72% of a 40 GiB swap was in use, which we did not investigate.

What survives is narrow: lazy unmount converts a visible problem into an invisible one, the object's eventual teardown or side effects land elsewhere, and in both cases nothing alerted on the cause. Section 5 sets out which claims remain.

1. A Command That Reports Success

A lazy unmount waits for nothing that holds a reference. Open files, other mount namespaces (the per-process or per-service views of the mount table that systemd creates for confined

services), dirty pages and the transport's own retry loop all count as references. The path stops resolving to the filesystem, the command exits 0, and the kernel carries on holding a superblock (the in-kernel object that represents a mounted filesystem) that nobody can name.

Our fleet powers off some machines on purpose, to save energy and heat. The control-plane and desktop host (64 cores, four GPUs, 503.1 GiB of RAM) mounts storage served by them. A small reconciler script, run at boot and from a five-minute timer, pings every node, mounts remote filesystems for nodes that answer, and unmounts those whose node is down. Before 23 June it unmounted lazily. Both incidents followed that pattern, one through the script and one through a human following the same sequence; only the second is shown to depend on it. "Zombie" here means a mount, distinct from the zombie client session in our March node-loss post, which was a Ceph session.

The evidence is retrospective. The June journal and activity record no longer exist, so we reconstructed that day from the long-term metrics store. The panic's crash dump was not available to us, so the kernel messages we quote from it are second-hand, as recorded in our notes at the time.

2. The Silent Freeze, 23 June

The node was power-cycled by the operator at 07:53 BST that morning (UTC is BST minus one hour; we use UTC for metrics and BST for the journal). At 08:25 both engineering nodes answered pings. At about 08:29 both were unreachable with chassis power read as off, a deliberate shutdown for heat management. One of them served an NFS-over-RDMA scratch volume (NFS carried over a remote direct memory access transport) that the compute node had mounted with sixteen connections, soft semantics and a short timeout, as recorded in our notes and not re-verified, because the mount table no longer holds that mount.

From 08:29 until the end, the reconciler lazily unmounted the dead mount, on the evidence of the commit message; the first unmount time is not recorded. Our notes record the kernel logging "server not responding" 3,135 times over about five hours; the journal that held those lines is gone, so the count is second-hand. The last journal line, at 13:34:54 BST, was the second of two "timed out" messages, and the journal ends cleanly: no panic, no out-of-memory, no lockup, no machine-check. The operator hard-reset the node at about 13:38. The interval from the engineering nodes going dark to the end of the journal was 5 h 05 m.

The notes written that day give a clean chain. The reconciler lazily unmounted the mount; the namespace entry vanished while the RDMA transport kept retrying; dirty pages were pinned behind it and could not be reclaimed. An hourly read-heavy corpus copy then began at about 13:20 BST, ballooned the page cache, and the kernel livelocked in direct reclaim (the path where an allocating task must reclaim memory synchronously). No out-of-memory kill, because reclaim never ended; no panic, because nothing crashed. We now think half of this is wrong.

The fleet's Prometheus store holds node memory counters at a two-minute step. The node was last scraped at 13:34:36 BST, and the window that matters is four samples.

UTC	MemFree GiB	MemAvailable GiB	Cached GiB	AnonPages GiB	Dirty MiB	Writeback
12:28	189.4	435.2	205.0	45.9	72	0
12:30	39.0	408.8	326.7	71.0	85	0
12:32	17.7	389.6	328.5	91.0	407	0
12:34	28.9	84.4	50.7	392.5	258	0

Table 1. Node memory counters in the last four scrapes before the June freeze. GiB values are bytes divided by 2^30; total memory is 503.1 GiB. AnonPages is private memory belonging to

processes.

Free memory had been flat, between 206.8 GiB at 11:00 and about 189 GiB, until 12:28 UTC. It then fell by about 172 GiB in four minutes. Private process memory sat near 43 GiB from 12:00 to 12:20 and at 45.9 GiB at 12:28, then reached 71, 91 and 392.5 GiB, a rise of a factor of 8.5 in about six minutes, with 302 GiB of it in the final two-minute interval.

Page cache did grow, from 205.0 to 328.5 GiB between 12:28 and 12:32, which a read-heavy copy would explain. But anonymous memory was growing in the same interval, and then the cache collapsed to 50.7 GiB at 12:34, which shows the kernel discarding clean cache as fast as it could to feed the anonymous growth. This is memory exhaustion with no swap to absorb it.

Three observations cut against the pinned-dirty-pages story. Dirty memory never exceeded 0.4 GiB. Writeback was zero at every sample from 12:24 to 12:34 except 4,096 bytes at 12:26 (the node was not scraped after 12:34, and later values are carried-forward copies), and the unstable-NFS-pages counter was zero at 12:34. The kernel's out-of-memory kill counter was zero at every real sample from 12:26 to 12:34.

Where the memory went is partly known. Pod working sets on the node were all under 100 MB at 12:34, and the Kubernetes agent service held 238 MB, while the root cgroup's resident set read 450.9 GB (420 GiB). The surge was in host-level processes outside the pods. Which process is not recorded and cannot now be recovered. The signature, hundreds of GiB of private memory in minutes with no out-of-memory kill, resembles the pattern in our July post on a 178 GiB agent memory leak, on the same class of host. A conspicuous anomaly, five hours of log lines, absorbed the blame for an inconspicuous one.

The kernel also exposes pressure stall information (PSI): cumulative time during which tasks were stalled, or waiting, on memory. These counters tell the same late story.

Counter	12:30	12:32	12:34
Stalled total (s)	0.0910	0.0939	3.2645
Waiting total (s)	0.0916	0.0946	3.2944

Table 2. Cumulative memory pressure counters. Before 12:30 both grew by under 0.001 s per two minutes.

Reclaim stalls appear only in the last interval: about 3.17 s of stalled time across two minutes, an average stalled rate of 2.64% and a waiting rate of 2.67% over 12:32 to 12:34.

The old memory alert fires at 90% of RAM used, measured on available memory. Available memory read 435.2, 408.8, 389.6 and then 84.4 GiB, which is 83.2% used at the last sample: alarming in hindsight, and still under threshold. The notes' reason for the silence (available memory near 408 GiB) holds only to 12:32. We then compared the three alerts written after the freeze against the data they were written for.

Signal	Alert threshold	Observed, last interval
Available memory used	over 90%	83.2% at 12:34 UTC
PSI stalled rate over 2 min	over 30% for 2 min	2.64% (2-min average, 12:32 to 12:34)
PSI waiting rate over 5 min	over 10% for 10 min	2.67% (2-min average, 12:32 to 12:34; the rule's 5-min rate would be lower)
Writeback	over 256 MB for 15 min	0

Table 3. Alert thresholds against the measurements from the June freeze. The two PSI rows and the writeback row are the alerts written after the freeze.

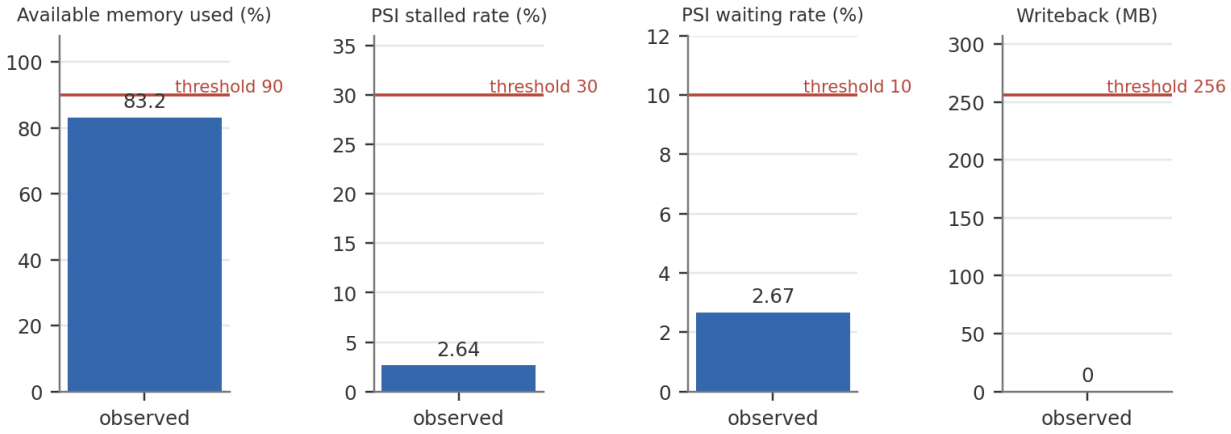


Figure 1. Each alert signal against its threshold, using the last measured interval before the June freeze. None of the three alerts added after the freeze would have fired on this data.

None of the three would have fired. The node died within about five minutes of the surge, so the "for" clauses alone make firing impossible for the waiting and writeback rules. The alerts we wrote were calibrated to the pinned-pages story, so they watch for slow, sustained stalls and undrained writeback, while the node died of a fast anonymous surge with writeback at zero.

3. The Kernel Panic, 2 November

A planned maintenance window powered off both engineering nodes for under an hour. Beforehand, GPU services were consolidated onto one GPU on the compute node, and its routes to the storage hosts were deleted from the fast private network so that Ceph traffic would use the ordinary network. Since 30 October the backup mount on the compute node had been a kernel CephFS client with `recover_session=no`, chosen because the alternative let an eviction silently discard dirty data while `fsync` and `rename` still returned success. Two long-running SQLite replication writers targeted that mount. The client had never been recovered from a real eviction in production before this day.

Time (UTC)	Event
About 06:55	Metadata-server session goes stale
06:59	Session hangs; stat, df and reads on the mount block in uninterruptible sleep
07:03	Reconciler logs "already mounted, OK" because its probe blocks instead of failing
07:10:10	Metadata server denies reconnect (from our records)
07:10:14 to 07:10:18	Client blocklisted
07:10:18	Operator-approved shell session runs a forced unmount (EBUSY), a lazy unmount, then a fresh mount
07:11:56	Kernel panic
07:12:19 to 07:42:46	Crash-capture kernel boots a full userland
08:38	Next normal boot

Table 4. Timeline of the November incident. Our records give three blocklist times (about 07:10, 07:10:14 and 07:10:18); we report the range.

The probe at 07:03 is the mirror image of June: it said mounted about a mount that served nothing, because the probe hung rather than failed and the script read a hang as health.

The mechanism below is from the crash dump's kernel log as quoted in our notes; we did not read the dump ourselves. The forced unmount failed with EBUSY because processes were stuck

in uninterruptible sleep on the mount. The lazy unmount then detached the old superblock from the visible namespace, and a new mount appeared at the same path. From the operator's seat, everything looked recovered.

The old superblock was not gone. Copies of the mount existed in the private mount namespaces of systemd-confined services, and they kept it alive. About 98 seconds later a process holding one such copy exited; it was not root (an ordinary user), since the unmount sequence, which did need root, had already run. Its exit path ran through the kernel's task exit and mount-cleanup routines into the CephFS superblock kill function, with inodes still busy. The recorded messages are "ceph: umount was killed, -512" and "VFS: Busy inodes after unmount of ceph", followed by a general protection fault at address 0xdead000000000165. That is a list-poison value, which the kernel writes into freed list entries: the usual signature of a use-after-free. A CephFS writeback worker then faulted dereferencing the freed superblock, and the kernel panicked. Our notes thus record a use-after-free in superblock teardown, inferred from the poison address and second-hand.

We checked the node's own system activity record for the hour, at ten-minute samples.

Time (UTC)	Free GiB	Available GiB	Dirty GiB	Memory used %	Swap used GiB
06:30	195.2	433.0	0.15	9.85	29.0
06:40	164.5	434.3	1.25	9.54	28.8
06:50	107.1	428.5	3.19	10.69	28.8
07:00	182.1	422.7	1.90	11.86	27.3
07:10	183.9	425.7	1.90	11.25	27.2

Table 5. Node memory from the system activity record on 2 November, kernel 7.0.0-38-generic, 64 CPUs. Converted from kilobyte columns. The node then had about 40 GiB of swap.

There was no shortage of available memory at ten-minute resolution. Swap was 68 to 72% used throughout, and we did not investigate why; swap had been added between June and November. Dirty memory held at 1.90 GiB across the last two samples, which is consistent with write-back stuck behind the evicted client, though we did not prove that.

The panic came from a manual run of a sequence that the reconciler also contained. After June we had rewritten the unmount helper to try a forced unmount first and fall back to lazy, both under a timeout, and to read the kernel's mount table instead of calling `mountpoint`, which hangs on a dead NFS mount. That was designed for NFS and was unsafe for CephFS. Since 30 October the reconciler was also meant to remount an evicted kernel client with the same sequence, so the timer path would have run it on the next eviction.

4. One Precondition, Two Different Failures

Feature	June	November
Remote filesystem	NFS over RDMA	Kernel CephFS
Server state	Powered off by design	Alive; client evicted after a route change
Action	Lazy unmount by the five-minute reconciler	Forced then lazy unmount by an operator session
Name in namespace afterwards	Gone	Gone (fresh mount at the same path)
Answer to "is it mounted"	No	Yes (the new mount)
Failure	Silent freeze of the node	Kernel panic

Feature	June	November
Delay to failure	5 h 05 m from the servers going dark (upper bound on unmount-to-failure)	98 s from the unmount command

Table 6. The two incidents compared. The June delay runs from the engineering nodes going dark (08:29 BST) to the end of the journal (13:34 BST). The two delays measure different things and are not like-for-like.

The lesson the data supports is small and mechanical. After a lazy unmount, the mount table and every probe built on it lose the object while the kernel keeps it, and its eventual teardown or its side effects land on an unrelated thread at an unrelated time.

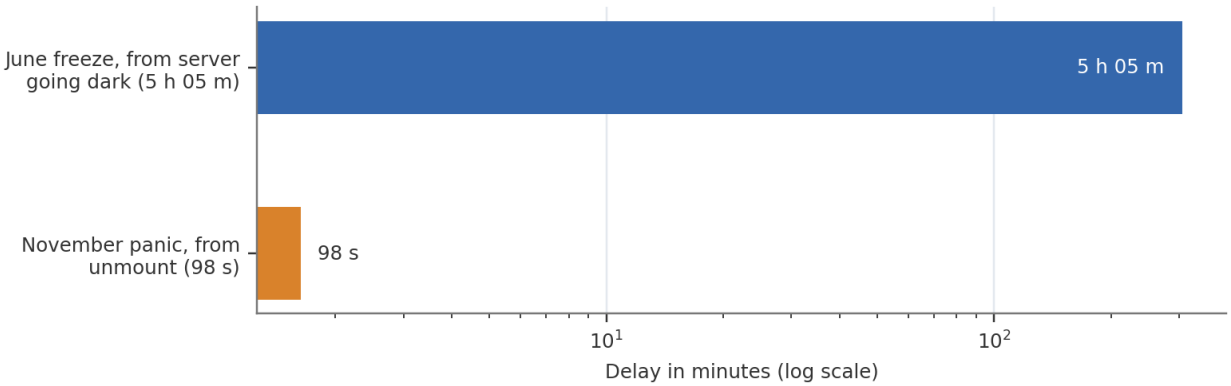


Figure 2. Delay to node failure. The June bar runs from the servers going dark, is an upper bound on any causal delay, and measures a different event from the November bar, which runs from the unmount command; the two are not comparable like for like.

5. What These Numbers Will Not Carry

There are two incidents on one host and no controlled reproduction. Nothing here is a rate.

For June, the evidence is Prometheus at two-minute resolution. The node stopped reporting after 13:34:36 BST, so the final minute is invisible and a spike inside the last interval could exceed the averages we computed. The June journal and activity record are gone, so the 3,135 log-line count, the mount options, the start time of the hourly copy and the lazy-only behaviour (which rests on a commit message) are second-hand. We use five hours and five minutes where our notes said five hours and eleven, because the audit record shows the engineering nodes reachable at 08:25 and unreachable at about 08:29.

What survives for June: the node froze silently, with no panic and no out-of-memory kill, within about five to seven minutes of the first sign of a memory surge (the 12:28 to 12:30 samples), a surge that available memory did not show until the last scrape, and a dead NFS mount had been handled by name-only unmounting for about five hours before that. What does not survive is the claim that the zombie mount caused the freeze, that dirty pages were pinned, or that a pinned-page reclaim livelock can be told apart from plain exhaustion by anything we hold. The unidentified allocator of about 300 GiB is the largest open question, along with whether the zombie contributed at all.

For November, the kernel messages are second-hand, and the number of stuck holders and the services that held private-namespace copies are not recorded. The system activity record has ten-minute resolution and no sample after 07:10, so a fast event in the final two minutes before the panic at 07:11:56 would be invisible. The 98-second gap is computed from two note timestamps (07:10:18 and 07:11:56); the reports say about 90 seconds. We did not reproduce the panic, so we cannot say whether it is a defect in the CephFS client, in the VFS mount-teardown path, or in the interaction of the unmount sequence with private mount

namespaces; the sequence needed root privileges, and we did not check whether the upstream kernel has changed the teardown. What survives is that a forced then lazy unmount of an evicted kernel CephFS client left a superblock alive in private namespaces, and a later process exit coincided with a recorded panic in its teardown, subject to the second-hand caveat.

6. What We Changed

On the day of the June freeze we changed the reconciler to force first and use lazy only as a fallback, with timeouts on both; to read mounts from the kernel's table; and to remove an embedded secret from the script. We added three alerts on memory pressure and writeback, which Section 2 shows would not have fired.

After the November panic we recorded a rule: never lazily unmount a kernel CephFS mount that has stuck holders. A forced unmount shuts the superblock down, the holders receive ESTALE or EIO and unblock, and then a plain unmount and remount can proceed. Before a planned outage, remount the backup mount from the ordinary network address while the fast path still works. The next experiment is the one we could not run: reproduce the November teardown on a test host with a deliberately blocklisted client and private-namespace holders, and measure whether any lazy-unmount variant is safe for a kernel CephFS mount. A hardware watchdog and a log alert on "not responding" rates, recommended after June, remain open. Swap now exists on the node, though the June surge was far larger than 40 GiB.

7. Methods and Artefacts

We used the fleet's long-term Prometheus store at a two-minute step; the node's system activity report for 2 November (kernel 7.0.0-38-generic, 64 CPUs, about 40 GiB of swap); the persistent journal listing (the earliest retained boot begins 31 October); the version-control history of the reconciler script and of the alert rules, first committed on 23 June at 14:08 BST after the freeze, so the pre-fix code is not retained; and our audit agent's per-round reports for 23 June. Statistics are unit conversions (bytes to GiB by 2^30) and two-point rates. The host has four GPUs, 64 cores and 540,149,735,424 bytes of memory, and had no swap in June. We retain the Prometheus series, the activity report, the script history and the session reports and notes.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-062. The metrics were reconstructed from the retained series after the 2 November incident; raw artefacts are retained as described in Section 7.