

MinIO Tiering Warning: Data Loss and Fault Tolerance Issues

Julien Laurenceau | Published 2025-12-06 | Updated 2026-09-30

<https://pepitedata.com/minio-tiering-bug-critical-data-loss/>

PepiteData | LinkedIn: Julien Laurenceau | X: @Julien_Lau



Update, September 2026: the sibling failure hit site replication. ILM expiration on a replicated mesh created delete markers that MinIO re-probes forever: 706 million replication requests in 48 hours, 96.8 percent answered 405, no data moved. Full write-up: [how 684 million HTTP 405s exposed the delete-marker replication loop](#).

Introduction

In the world of big data, storage solutions are the backbone of any architecture. As an expert solutions architect, I have relied on various storage solutions to ensure data integrity and availability. Recently, I discovered a critical flaw in MinIO's tiering feature, introduced in RELEASE.2022-11-10T18-20-21Z and still present in version 2024-09-22T00:33:43Z, that poses a significant risk to data integrity. This feature is inspired by [AWS S3 lifecycle transition](#)

MinIO is very simple to use, which is why it is frequently chosen for testing purposes to simulate S3 in CI/CD pipelines. In these cases, data loss is not an issue. On the other hand, MinIO is also highly appreciated in on-premises environments for providing an S3 API that is very similar to AWS S3 and much simpler to set up than the alternative Ceph + Rados Gateway. In such cases, MinIO is often a

critical part of the enterprise data lake and is operated with very high resilience and durability targets. Data loss is not tolerable.

Note: This is an ongoing investigation, and I have not received any insights from the MinIO team. However, I experienced the issue on a production cluster and was able to reproduce it in a test setup. Two different versions of MinIO were tested without success.

About tiering

Tiering moves cold objects to remote storage and leaves a stub plus metadata behind on the hot tier. All requests still hit the hot tier, transitions run in daily batches, and the delay cannot be shorter than one day. The upstream documentation describes the feature; the failure this post covers lives inside it.

[Minio doc on ILM](#)

Object Transition (“Tiering”)

MinIO supports creating object transition lifecycle management rules, where MinIO can automatically move an object to a remote storage “tier”. MinIO supports any of the following remote tier targets:

- [MinIO](#)
- [Amazon S3](#)
- [Google Cloud Storage](#)
- [Microsoft Azure Blob Storage](#)

Important:

MinIO Object Transition supports cost-saving strategies around moving older or aged data to cost-optimized remote storage tiers, such as cloud storage or high-density HDD storage.

Important Facts About the Tiering Feature:

Tiering behaves nothing like a cache, and the properties below explain the failure that follows. All requests, reads included, hit the hot tier. Transitions run in daily batches with a delay of at least one day. The hot tier serves every metadata operation, and small objects live inside the metadata file itself.

- Once tiering is configured, all requests must go to the unique hot tier.
- Backend tier(s) cannot be used, even for read-replica purposes. Under the hood, MinIO uses UUIDs to name transitioned objects.
- The hot tier holds the metadata of all objects (from all tiers). As such, there are no lookups to cold tiers on LIST requests. This is particularly powerful for on-premise setups, where one can have a small SSD cluster as the front tier and a large HDD cluster as the cold tier, achieving great performance.
- The transition strategy is dictated by AWS algorithms, and transitions are made in daily batches starting at 00:00 UTC.
- The transition delay cannot be shorter than one day.

More details about the philosophy around the tiering feature of Minio in [github issue 18821](#)

The Issue at Hand

MinIO's tiering feature was designed to optimize storage by transitioning objects between different storage classes. However, enabling this feature can lead to severe data loss. The root of the problem lies in the MinIO scanner's inability to repair metadata of transitioned objects. This flaw is akin to a lack of anti-entropy algorithm (hello Cassandra friends !), where data consistency cannot be repaired once compromised.

The Consequences

Without the ability to repair metadata, every outage or drive replacement risks losing quorum. The tiering variant is worse: a transitioned object whose metadata disappears from two drives vanishes from listings while its data sits intact and unreachable on the cold tier. Over time this leads to data loss, and for organizations relying on MinIO for critical data the consequences are catastrophic.

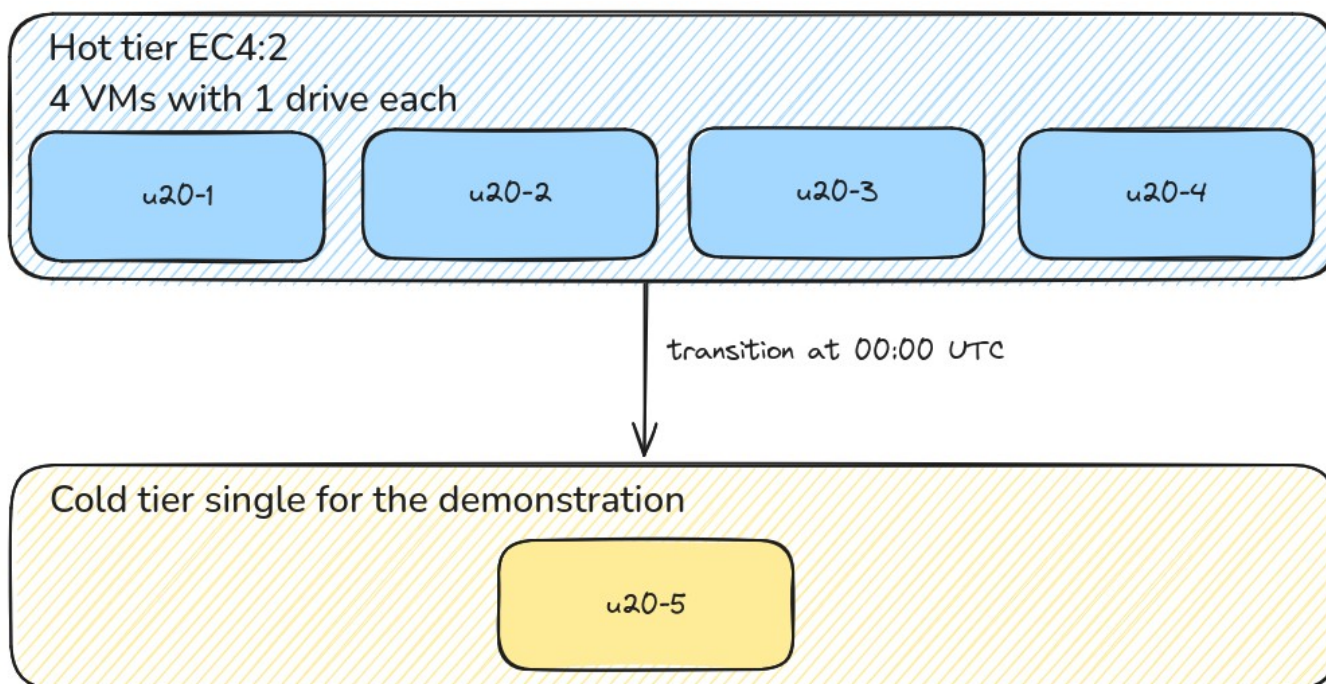
Technical Details

The mechanics of the failure, as [documented on GitHub \(Issue #20559\)](#), show which files hold the metadata, which repair paths exist and where each path gives up. The gap is structural rather than a tuning mistake, so no configuration on an affected release closes it. What follows is the technical record.

MinIO uses Erasure Coding to ensure fault tolerance. Without tiering, replacing faulty drives is a routine operation. Once a faulty drive is replaced, MinIO immediately detects that it is empty and initiates its healing process to rebuild the data.

Recently, I had to replace a faulty server in an on-premise cluster using tiering. To my surprise, many users complained about LIST consistency issues, which prompted this analysis.

The smallest test setup involves 5 virtual machines (VMs). The hot tier needs to effectively use erasure coding, requiring a minimum cluster size of 4 nodes. The cold tier only needs to exist to configure the transition, so I used a single VM, although it could have been a path to an AWS S3 bucket.



These two terminal captures show the version used and the files stored in the test bucket named "buc." As can be seen, all objects from the directory "adir" transitioned to the cold tier. We can still find the

metadata of all these transitioned objects by searching for the "xl.meta" files directly in the MinIO drives.

Version: 2024-10-13T13:34:11Z
Network: 4/4 OK
Drives: 1/1 OK
Pool: 1

Pool	Drives Usage	Erasure stripe size	Erasure sets
1st	1.5% (total: 8.0 GiB)	4	1

2.8 MiB Used, 1 Bucket, 21 Objects
4 drives online, 0 drives offline, EC:2

~/ [jlu@u20-4]>

6: Terminal ▾

~/ [jlu@u20-5]> mc admin info local | tail -12
Version: 2024-10-13T13:34:11Z
Network: 1/1 OK
Drives: 1/1 OK
Pool: 1

Pool	Drives Usage	Erasure stripe size	Erasure sets
1st	1.6% (total: 4.0 GiB)	1	1

2.7 MiB Used, 1 Bucket, 11 Objects
1 drive online, 0 drives offline, EC:0

~/ [jlu@u20-5]>

Pool	Drives Usage	Erasure stripe size	Erasure sets
1st	1.5% (total: 8.0 GiB)	4	1

2.8 MiB Used, 1 Bucket, 21 Objects
4 drives online, 0 drives offline, EC:2

~/ [jlu@u20-4]> mc ilm tier ls local

Name	Type	Endpoint	Bucket	Prefix	Region	Storage-Class
MINIO-COLD-BUC	minio	http://vm5:9000	buc	/	-	-

~/ [jlu@u20-4]>

6: Terminal ▾

Pool	Drives Usage	Erasure stripe size	Erasure sets
1st	1.6% (total: 4.0 GiB)	1	1

2.7 MiB Used, 1 Bucket, 11 Objects
1 drive online, 0 drives offline, EC:0

~/ [jlu@u20-5]> mc ilm tier ls local

Name	Type	Endpoint	Bucket	Prefix	Region	Storage-Class

~/ [jlu@u20-5]>

```
[jlu@u20-1]~> mc ls -r local/buc/
[2024-10-28 20:12:20 CET] 200KiB MINIO-COLD-BUC adir/testfile_241028_idx20
[2024-10-28 20:12:20 CET] 210KiB MINIO-COLD-BUC adir/testfile_241028_idx21
[2024-10-28 20:12:20 CET] 220KiB MINIO-COLD-BUC adir/testfile_241028_idx22
[2024-10-28 20:12:20 CET] 230KiB MINIO-COLD-BUC adir/testfile_241028_idx23
[2024-10-28 20:12:20 CET] 240KiB MINIO-COLD-BUC adir/testfile_241028_idx24
[2024-10-28 20:12:20 CET] 250KiB MINIO-COLD-BUC adir/testfile_241028_idx25
[2024-10-28 20:12:20 CET] 260KiB MINIO-COLD-BUC adir/testfile_241028_idx26
[2024-10-28 20:12:20 CET] 270KiB MINIO-COLD-BUC adir/testfile_241028_idx27
[2024-10-28 20:12:20 CET] 280KiB MINIO-COLD-BUC adir/testfile_241028_idx28
[2024-10-28 20:12:20 CET] 290KiB MINIO-COLD-BUC adir/testfile_241028_idx29
[2024-10-28 20:12:20 CET] 300KiB MINIO-COLD-BUC adir/testfile_241028_idx30
[2024-10-29 14:03:29 CET] 11KiB STANDARD bdir/testfile_241029_idx11
[2024-10-29 14:03:29 CET] 12KiB STANDARD bdir/testfile_241029_idx12
[2024-10-29 14:03:29 CET] 13KiB STANDARD bdir/testfile_241029_idx13
[2024-10-29 14:03:29 CET] 14KiB STANDARD bdir/testfile_241029_idx14
[2024-10-29 14:03:29 CET] 15KiB STANDARD bdir/testfile_241029_idx15
[2024-10-29 14:03:29 CET] 16KiB STANDARD bdir/testfile_241029_idx16
[2024-10-29 14:03:29 CET] 17KiB STANDARD bdir/testfile_241029_idx17
[2024-10-29 14:03:29 CET] 18KiB STANDARD bdir/testfile_241029_idx18
[2024-10-29 14:03:29 CET] 19KiB STANDARD bdir/testfile_241029_idx19
[2024-10-29 14:03:29 CET] 20KiB STANDARD bdir/testfile_241029_idx20
[jlu@u20-1]~> mc admin heal -r --scan=deep -a --force local/buc
🟡 buc/bdir/testfile_241029_idx20
0/21 objects; 155 KiB in 1s
```

Green	14	53.8%	
Yellow	0	0.0%	
Red	0	0.0%	
Grey	11	42.3%	

There are 21 objects in this bucket "buc" separated in 2 directories (adir: 11 objects cold + bdir: 10 objects hot), but the the scanner sees :

- 11 grey objects : corresponding to the metadata file of transitioned objects. 1 metadata file per object.
- 14 green objects : for sure 10 of these objects are the recent objects in the the "STANDARD" hot tier. The other 4 objects found is a non-sense.

Sidenote: Minio uses data inlining of small objects: in order to prevent loosing too much IOPS on small files, if the data size is smaller than 256KiB, then Minio does not create a datafile and the data is inlined in the metadata file "xl.meta".

In the next step, the file named "testfile_241028_idx20" is directly removed from the MinIO drive of the node u20-1 to simulate a faulty drive. In a non-tiered setup, the healing would be immediate upon the next read of this object. In this example, I forced a heal and read this object, but as can be seen, the "xl.meta" was never healed on u20-1. At this stage, there is still no data loss.

```
[jlu@u20-1]# f='testfile_241028_idx20'
[jlu@u20-1]# mc ls local/buc/adir/$f
[2024-10-28 20:12:20 CET] 200KiB MINIO-COLD-BUC testfile_241028_idx20
[jlu@u20-1]# find /data/1/buc -type f | grep $f
/data/1/buc/adir/testfile_241028_idx20/xl.meta
[jlu@u20-1]# ll /data/1/buc/adir/$f/xl.meta
-rw-r--r-- 1 minio-user minio-user 686 oct. 30 01:09 /data/1/buc/adir/testfile_241028_idx20/xl.meta
[jlu@u20-1]# da
da: command not found
[jlu@u20-1]# date
mer. 30 oct. 2024 12:30:58 CET
[jlu@u20-1]# for i in range {1..10}; do printf "$(mc ls -r local/buc/ | wc -l) "; done; echo ""
21 21 21 21 21 21 21 21 21 21
[jlu@u20-1]# mc ls local/buc/adir/$f; date; sudo systemctl stop minio; date; sudo rm -f $(find /data/1/buc -type
f | grep $f); ll /data/1/buc/adir/$f/xl.meta; sudo systemctl start minio; date; ll /data/1/buc/adir/$f/xl.meta; mc
ls local/buc/adir/$f;
[2024-10-28 20:12:20 CET] 200KiB MINIO-COLD-BUC testfile_241028_idx20
mer. 30 oct. 2024 12:31:48 CET
mer. 30 oct. 2024 12:31:48 CET
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
mer. 30 oct. 2024 12:31:49 CET
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
[jlu@u20-1]# ll /data/1/buc/adir/$f/xl.meta
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
[jlu@u20-1]# date
mer. 30 oct. 2024 12:32:12 CET
[jlu@u20-1]# for i in range {1..10}; do printf "$(mc ls -r local/buc/ | wc -l) "; done; echo ""
21 21 21 21 21 21 21 21 21 21
[jlu@u20-1]# mc admin info local | tail -2
2.8 MiB Used, 1 Bucket, 21 Objects
4 drives online, 0 drives offline, EC:2
[jlu@u20-1]# find /data/1/buc -type f | grep $f
[jlu@u20-1]# mc admin heal -r --scan=deep -a --force local/buc
buc/bdir/testfile_241029_idx20
0/21 objects; 155 KiB in 1s

Green 14 53.8%
Yellow 0 0.0%
Red 0 0.0%
Grey 11 42.3%
```

In the next step, I performed the same action on the node u20-2 and observed a catastrophic failure: data was lost.

```
[2024-10-28 20:12:20 CET] 200KiB MINIO-COLD-BUC testfile_241028_idx20
[jlu@u20-2]# date
mer. 30 oct. 2024 12:41:43 CET
[jlu@u20-2]# find /data/1/buc -type f | grep $f
/data/1/buc/adir/testfile_241028_idx20/xl.meta
[jlu@u20-2]# ll /data/1/buc/adir/$f/xl.meta
-rw-r--r-- 1 minio-user minio-user 686 oct. 30 01:09 /data/1/buc/adir/testfile_241028_idx20/xl.meta
[jlu@u20-2]# mc admin heal -r --scan=deep -a --force local/buc
buc/bdir/testfile_241029_idx20
0/21 objects; 155 KiB in 1s

Green 14 53.8%
Yellow 0 0.0%
Red 0 0.0%
Grey 11 42.3%
```

```
[jlu@u20-2]# for i in range {1..10}; do printf "$(mc ls -r local/buc/ | wc -l) "; done; echo ""
21 21 21 21 21 21 21 21 21 21
[jlu@u20-2]# mc ls local/buc/adir/$f; date; sudo systemctl stop minio; date; sudo rm -f $(find /data/1/buc -type
f | grep $f); ll /data/1/buc/adir/$f/xl.meta; sudo systemctl start minio; date; ll /data/1/buc/adir/$f/xl.meta; mc
ls local/buc/adir/$f;
[2024-10-28 20:12:20 CET] 200KiB MINIO-COLD-BUC testfile_241028_idx20
mer. 30 oct. 2024 12:42:15 CET
mer. 30 oct. 2024 12:42:16 CET
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
[2024-10-28 20:12:20 CET] 200KiB MINIO-COLD-BUC testfile_241028_idx20
[jlu@u20-2]# for i in range {1..10}; do printf "$(mc ls -r local/buc/ | wc -l) "; done; echo ""
21 21 21 21 21 21 21 21 21 21
[jlu@u20-2]# mc ls local/buc/adir/$f
[2024-10-28 20:12:20 CET] 200KiB MINIO-COLD-BUC testfile_241028_idx20
[jlu@u20-2]# ll /data/1/buc/adir/$f/xl.meta
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
[jlu@u20-2]# mc stat local/buc/adir/$f
mc: <ERROR> Unable to stat 'local/buc/adir/testfile_241028_idx20'. Object does not exist.
[jlu@u20-2]# mc cp local/buc/adir/$f /tmp/
mc: <ERROR> Unable to prepare URL for copying. Unable to guess the type of copy operation.
[jlu@u20-2]# mc admin heal -r --scan=deep -a --force local/buc
buc/bdir/testfile_241029_idx20
0/21 objects; 155 KiB in 1s

Green 15 57.7%
Yellow 0 0.0%
Red 0 0.0%
Grey 10 38.5%
```

```
[jlu@u20-2]# find /data/1/buc -type f | grep $f
[jlu@u20-2]# ll /data/1/buc/adir/$f/xl.meta
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
[jlu@u20-2]# mc ls local/buc/adir/$f
[2024-10-28 20:12:20 CET] 200KiB MINIO-COLD-BUC testfile_241028_idx20
[jlu@u20-2]# mc cp local/buc/adir/$f /tmp/
mc: <ERROR> Unable to prepare URL for copying. Unable to guess the type of copy operation.
[jlu@u20-2]# mc stat local/buc/adir/$f
mc: <ERROR> Unable to stat 'local/buc/adir/testfile_241028_idx20'. Object does not exist.
[jlu@u20-2]# mc admin info local | tail -2
2.8 MiB Used, 1 Bucket, 21 Objects
4 drives online, 0 drives offline, EC:2
[jlu@u20-2]#
```

At this stage, I am still able to list the object despite using the MinIO setting "list_quorum=strict." However, I am no longer able to read the data.

After performing the same action on u20-3, the data is completely gone, and I am not even able to list the lost object. If you do not use a delta table or some other catalog, you might never notice that data was lost.

```
[2024-10-28 20:12:20 CET] 268KiB MINIO-COLD-BUC adir/testfile_241028_idx26
[2024-10-28 20:12:20 CET] 278KiB MINIO-COLD-BUC adir/testfile_241028_idx27
[2024-10-28 20:12:20 CET] 288KiB MINIO-COLD-BUC adir/testfile_241028_idx28
[2024-10-28 20:12:20 CET] 298KiB MINIO-COLD-BUC adir/testfile_241028_idx29
[2024-10-28 20:12:20 CET] 308KiB MINIO-COLD-BUC adir/testfile_241028_idx30
[2024-10-29 14:03:29 CET] 11KiB STANDARD bdir/testfile_241029_idx11
[2024-10-29 14:03:29 CET] 12KiB STANDARD bdir/testfile_241029_idx12
[2024-10-29 14:03:29 CET] 13KiB STANDARD bdir/testfile_241029_idx13
[2024-10-29 14:03:29 CET] 14KiB STANDARD bdir/testfile_241029_idx14
[2024-10-29 14:03:29 CET] 15KiB STANDARD bdir/testfile_241029_idx15
[2024-10-29 14:03:29 CET] 16KiB STANDARD bdir/testfile_241029_idx16
[2024-10-29 14:03:29 CET] 17KiB STANDARD bdir/testfile_241029_idx17
[2024-10-29 14:03:29 CET] 18KiB STANDARD bdir/testfile_241029_idx18
[2024-10-29 14:03:29 CET] 19KiB STANDARD bdir/testfile_241029_idx19
[2024-10-29 14:03:29 CET] 20KiB STANDARD bdir/testfile_241029_idx20
[jlu@u20-3]~$ f='testfile_241028_idx20'
[jlu@u20-3]~$ mc ls local/buc/adir/$f
[2024-10-28 20:12:20 CET] 288KiB MINIO-COLD-BUC testfile_241028_idx20
[jlu@u20-3]~$ find /data/1/buc -type f | grep $f
/data/1/buc/adir/testfile_241028_idx20/xl.meta
[jlu@u20-3]~$ ll /data/1/buc/adir/$f/xl.meta
-rw-r--r-- minio-user minio-user 686 oct. 30 01:09 /data/1/buc/adir/testfile_241028_idx20/xl.meta
[jlu@u20-3]~$ for i in range(1..10); do printf "%(mc ls -r local/buc/ | wc -l) "; done; echo ""
21 21 21 21 21 21 21 21 21 21
[jlu@u20-3]~$ mc ls local/buc/adir/$f; date; sudo systemctl stop minio; date; sudo rm -f $(find /data/1/buc -type
f | grep $f); ll /data/1/buc/adir/$f/xl.meta; sudo systemctl start minio; date; ll /data/1/buc/adir/$f/xl.meta; mc
ls local/buc/adir/$f;
[2024-10-28 20:12:20 CET] 288KiB MINIO-COLD-BUC testfile_241028_idx20
mer. 30 oct. 2024 12:47:41 CET
mer. 30 oct. 2024 12:47:41 CET
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
mer. 30 oct. 2024 12:47:41 CET
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
[jlu@u20-3]~$ for i in range(1..10); do printf "%(mc ls -r local/buc/ | wc -l) "; done; echo ""
20 20 20 20 20 20 20 20 20 20
[jlu@u20-3]~$ mc admin info local | tail -2
2.8 MiB Used, 1 Bucket, 21 Objects
4 drives online, 0 drives offline, EC:2
[jlu@u20-3]~$ find /data/1/buc -type f | grep $f
[jlu@u20-3]~$ ll /data/1/buc/adir/$f/xl.meta
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
[jlu@u20-3]~$ mc stat local/buc/adir/$f
mc: <ERROR> Unable to stat 'local/buc/adir/testfile_241028_idx20'. Object does not exist.
[jlu@u20-3]~$ mc admin heal -r --scan=deep -a --force local/buc
o buc/bdir/testfile_241029_idx20
0/20 objects; 155 KiB in 1s
Green 14 56.0% ██████████
Yellow 0 0.0%
Red 0 0.0%
Grey 10 40.0% ██████████
[jlu@u20-3]~$ ll /data/1/buc/adir/$f/xl.meta
ls: cannot access '/data/1/buc/adir/testfile_241028_idx20/xl.meta': No such file or directory
[jlu@u20-3]~$ mc ls local/buc/adir/$f
[jlu@u20-3]~$ mc cp local/buc/adir/$f /tmp/
mc: <ERROR> Unable to prepare URL for copying. Unable to guess the type of copy operation.
[jlu@u20-3]~$ mc stat local/buc/adir/$f
mc: <ERROR> Unable to stat 'local/buc/adir/testfile_241028_idx20'. Object does not exist.
[jlu@u20-3]~$
```

Once you reach this point, there is no solution to heal the cluster. You are forced to use a backup and rewrite the data to the hot tier to ensure consistent metadata across all nodes.

This situation is even more frustrating because the data remains intact on the cold tier! However, since MinIO uses internal names and directory structures on the cold tier, it is impossible to heal the data using the cold tier as can ben seen below.

```

~/ [jlu@u20-5]~> mc ls -r local/buc
[2024-10-30 01:09:48 CET] 200KiB STANDARD 1c2f08876460347e/30/2d/302d58b6-8722-47e2-a4de-8be87a59d3b3
[2024-10-30 01:09:48 CET] 280KiB STANDARD 1c2f08876460347e/31/fd/31fdd0cc-3519-4e84-9632-d4051a2e8a09
[2024-10-30 01:09:48 CET] 300KiB STANDARD 1c2f08876460347e/40/f2/40f22c19-13bf-4841-babb-cc8b4389c883
[2024-10-30 01:09:48 CET] 240KiB STANDARD 1c2f08876460347e/54/72/5472f330-f21b-4e7f-a67b-d82cd6d067de
[2024-10-30 01:09:48 CET] 210KiB STANDARD 1c2f08876460347e/58/89/58892800-82e9-46e5-ae4f-5c11b3d20c90
[2024-10-30 01:09:48 CET] 260KiB STANDARD 1c2f08876460347e/63/d9/63d90874-3a8c-467d-b5d5-8b5d9fe2ded2
[2024-10-30 01:09:48 CET] 220KiB STANDARD 1c2f08876460347e/c8/46/c846a088-5c5a-4fee-a80f-ae9f3e08bdd9
[2024-10-30 01:09:48 CET] 290KiB STANDARD 1c2f08876460347e/c8/cf/c8cfc958-e04a-4243-8c4c-124c76d4a1f7
[2024-10-30 01:09:48 CET] 270KiB STANDARD 1c2f08876460347e/d7/40/d74022a5-fb91-4a42-80f1-668324f118dd
[2024-10-30 01:09:48 CET] 250KiB STANDARD 1c2f08876460347e/dd/74/dd74e121-2958-4cbb-a479-9bc4f9e60cd
[2024-10-30 01:09:48 CET] 230KiB STANDARD 1c2f08876460347e/f7/34/f734a820-4847-4f1a-bfc4-1fe5495502a6
~/ [jlu@u20-5]~> find /data/1/buc -type f -exec ls -la {} \;;
bash: syntax error near unexpected token `;'
~/ [jlu@u20-5]~> find /data/1/buc -type f -exec ls -la {} \;;
-rw-r--r-- 1 minio-user minio-user 245792 oct. 30 01:09 /data/1/buc/1c2f08876460347e/54/72/5472f330-f21b-4e7f-a67b-d82cd6d067de/81644c46-ea96-4f8f-b290-4b10c88376c0/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/54/72/5472f330-f21b-4e7f-a67b-d82cd6d067de/xl.meta
-rw-r--r-- 1 minio-user minio-user 266272 oct. 30 01:09 /data/1/buc/1c2f08876460347e/63/d9/63d90874-3a8c-467d-b5d5-8b5d9fe2ded2/cfff2452-7825-4396-bfc1-a7eb05846f0b/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/63/d9/63d90874-3a8c-467d-b5d5-8b5d9fe2ded2/xl.meta
-rw-r--r-- 1 minio-user minio-user 235552 oct. 30 01:09 /data/1/buc/1c2f08876460347e/f7/34/f734a820-4847-4f1a-bfc4-1fe5495502a6/f271a514-a8c7-46d5-b6ae-2574669e0491/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/f7/34/f734a820-4847-4f1a-bfc4-1fe5495502a6/xl.meta
-rw-r--r-- 1 minio-user minio-user 215072 oct. 30 01:09 /data/1/buc/1c2f08876460347e/58/89/58892800-82e9-46e5-ae4f-5c11b3d20c90/6f5271c5-f071-408c-abc7-b4144c8f227c/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/58/89/58892800-82e9-46e5-ae4f-5c11b3d20c90/xl.meta
-rw-r--r-- 1 minio-user minio-user 256032 oct. 30 01:09 /data/1/buc/1c2f08876460347e/dd/74/dd74e121-2958-4cbb-a479-9bc4f9e60cd/8ae7bde6-cdd6-4db2-a711-8354c89b2a26/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/dd/74/dd74e121-2958-4cbb-a479-9bc4f9e60cd/xl.meta
-rw-r--r-- 1 minio-user minio-user 276512 oct. 30 01:09 /data/1/buc/1c2f08876460347e/d7/40/d74022a5-fb91-4a42-80f1-668324f118dd/2760dec3-e89b-4e5d-8637-9f2f9cb2b6a3/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/d7/40/d74022a5-fb91-4a42-80f1-668324f118dd/xl.meta
-rw-r--r-- 1 minio-user minio-user 204832 oct. 30 01:09 /data/1/buc/1c2f08876460347e/30/2d/302d58b6-8722-47e2-a4de-8be87a59d3b3/13e431ee-5686-4953-aal1-1e3aa4edc7f3/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/30/2d/302d58b6-8722-47e2-a4de-8be87a59d3b3/xl.meta
-rw-r--r-- 1 minio-user minio-user 225312 oct. 30 01:09 /data/1/buc/1c2f08876460347e/c8/46/c846a088-5c5a-4fee-a80f-ae9f3e08bdd9/e6821b4a-483e-4247-81fa-573a7aae3932/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/c8/46/c846a088-5c5a-4fee-a80f-ae9f3e08bdd9/xl.meta
-rw-r--r-- 1 minio-user minio-user 296992 oct. 30 01:09 /data/1/buc/1c2f08876460347e/c8/cf/c8cfc958-e04a-4243-8c4c-124c76d4a1f7/e4df8926-6ff5-41a6-8e02-a88f9395eb93/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/c8/cf/c8cfc958-e04a-4243-8c4c-124c76d4a1f7/xl.meta
-rw-r--r-- 1 minio-user minio-user 387232 oct. 30 01:09 /data/1/buc/1c2f08876460347e/40/f2/40f22c19-13bf-4841-babb-cc8b4389c883/3c817c41-a53f-483a-9d4c-ebcc3b8daea0/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/40/f2/40f22c19-13bf-4841-babb-cc8b4389c883/xl.meta
-rw-r--r-- 1 minio-user minio-user 286752 oct. 30 01:09 /data/1/buc/1c2f08876460347e/31/fd/31fdd0cc-3519-4e84-9632-d4051a2e8a09/fbffa9ba-eca6-47d0-8080-67dde5ad6ca0/part.1
-rw-r--r-- 1 minio-user minio-user 393 oct. 30 01:09 /data/1/buc/1c2f08876460347e/31/fd/31fdd0cc-3519-4e84-9632-d4051a2e8a09/xl.meta

```

Shell snippet

The reproduction is one destructive step: remove the metadata file for a transitioned object on a second node and watch the heal refuse to bring it back. The snippet below is the sequence used in the five-VM test setup, and the same steps produced data loss on production.

```

rm testfile_*.idx*
for i in {1..20}; do dd if=/dev/urandom bs=1K count=$i of=testfile_$(date +%Y%m%d*)_idx$i;
done
mc cp testfile_*.idx* local/buc/adir/
mc ls -r local/buc/
mc admin info local | tail -12
date
f='testfile_241028_idx20'
mc ls local/buc/adir/$f
find /data/1/buc -type f | grep $f
ll /data/1/buc/adir/$f/xl.meta
mc admin heal -r --scan=deep -a --force local/buc
for i in range {1..10}; do printf "$(mc ls -r local/buc/ | wc -l) "; done; echo ""
mc ls local/buc/adir/$f; date; sudo systemctl stop minio; date ; sudo rm -f $(find /data/1/buc
-type f | grep $f); ll /data/1/buc/adir/$f/xl.meta; sudo systemctl start minio ; date ; ll
/data/1/buc/adir/$f/xl.meta; mc ls local/buc/adir/$f;
for i in range {1..10}; do printf "$(mc ls -r local/buc/ | wc -l) "; done; echo ""
mc admin info local | tail -2
ll /data/1/buc/adir/$f/xl.meta
mc stat local/buc/adir/$f
mc admin heal -r --scan=deep -a --force local/buc
ll /data/1/buc/adir/$f/xl.meta
mc ls local/buc/adir/$f
mc cp local/buc/adir/$f /tmp/
mc stat local/buc/adir/$f

```

Speculations

What follows goes beyond the recorded facts and is marked as speculation on purpose. The core inference is that broken healing tests and continuous releases compound: an untested repair path can rot for months while every release looks green. The evidence above is reproducible; this section is interpretation.

Firstly, bugs of this nature should not occur in a production grade storage solution. Their existence suggests that the developers did not adequately test the product.

Technically, the fact that the tiering feature uses scheduled batches rather than streaming complicates matters: you must wait (or manipulate the clock) until 00:00 UTC the next day for the transition to occur.

Regardless, resilience and durability tests must be conducted!

Grey	<i>Unrecoverable</i> , the object has lost too many data and/or parity shards and cannot be healed or recovered
-------------	---

Secondly, from an architectural perspective, I believe the tiering feature has a major flaw on the operationnal side. In MinIO, the anti-entropy mechanism is the "scanner" which runs continuously to perform a full scan of all objects on a regular basis. Additionally, there is a read-repair mechanism triggered when some shards are unavailable during read queries. This system works well when tiering is not used. However, when tiering is enabled, the hot tier only holds metadata of transitioned objects. Consequently, the scanner perceives these as "grey objects" which means "unrecoverable objects". This makes monitoring for data loss nearly impossible: you can no longer monitor your hot tier for data loss, as transitioned objects in normal state appear with the the same status as lost objects.

Thirdly, minio does not use semver and does not support rollback of versions. There is no point in trying to bissect versions in order to try to identify a non broken version.

Recommendations

Three recommendations follow directly from the reproduction: disable tiering on any cluster where data loss is unacceptable, keep independent backups because replication propagates the same corruption, and monitor release notes for a verified fix before re-enabling the feature. The longer form is below.

- **Disable Tiering:** Until a fix is released, consider disabling the tiering feature to prevent potential data loss.
- **Regular Backups:** Ensure that regular backups are in place to mitigate the risk of data loss.
- **Monitor Updates:** Keep an eye on MinIO's updates and patches for a resolution to these issues.

Conclusion

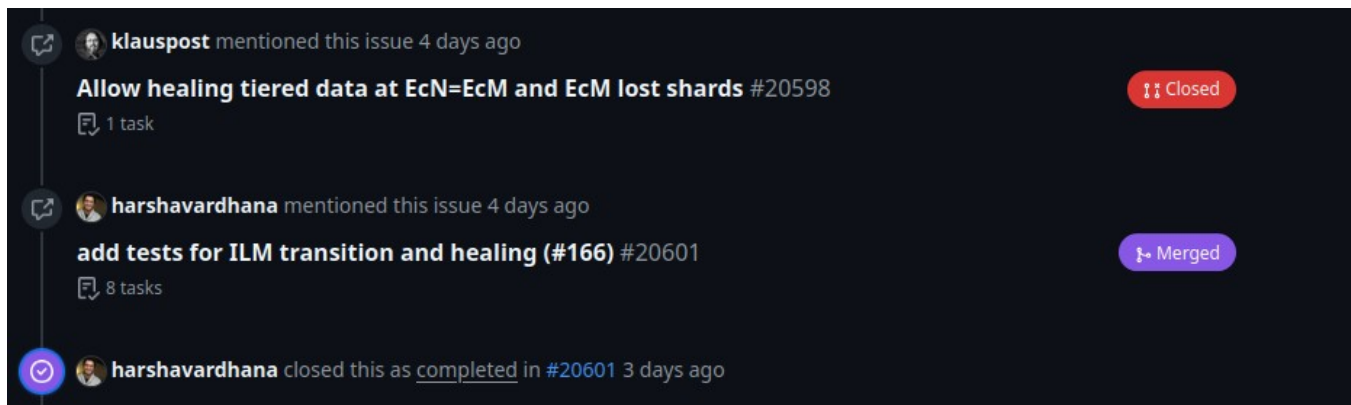
Raising awareness about this critical flaw is essential for the community. While MinIO has been a reliable storage solution, this issue underscores the importance of thorough testing and validation of new features. I urge MinIO to address this flaw promptly to try restore user confidence.

However, from my perspective Minio is not for production workloads where dataloss is not acceptable. Minio release process with no SemVer and non extensive tests is a problem. In addition, Minio does nothing to inform operator of potential data loss. In my case, this was made obvious only after loosing ~10% of our data and having a lot of client issues.

Edit November 4th 2024

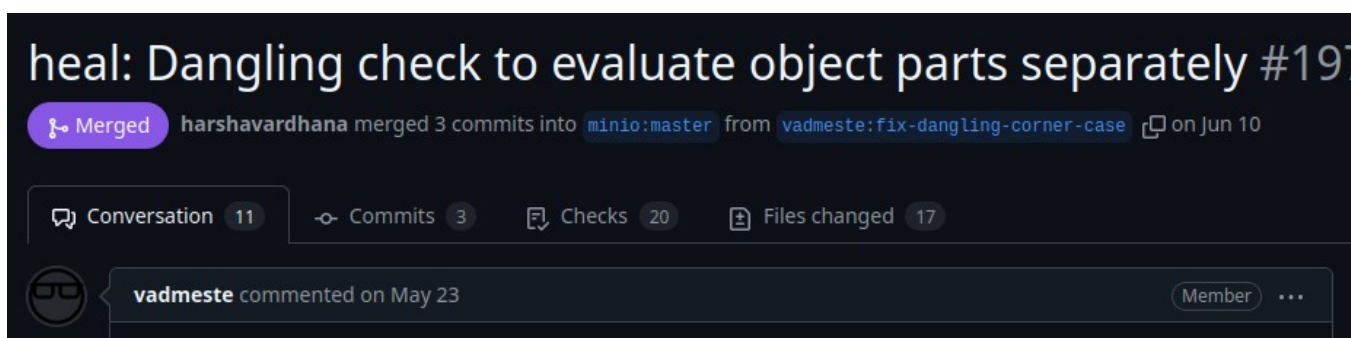
A few days after the initial post of this article, Minio promptly proposed a bugfix to this issue : thanks to them !

However, I did not test it yet since it is merged but not released in any official version yet. The issue on github is now closed and unfortunately I will not be able to share the results of my own tests there.



Scaring point : Minio tests for healing were known to be broken since more than 6 months ([opening of regression ticket 19797](#)). Despite that, minio continuous release cycle was not interrupted and multiple versions were rolled out to production without appropriate tests.

This is a red flag for production usage.



Edit November 15th 2024

MinIO released version RELEASE.2024-11-07T00-52-20Z and it changes two things. It fixes the healing of transitioned objects, the failure demonstrated above, and it repairs the internal testing suite that had been broken for months. The release note states both, in these words:

add tests for ILM transition and healing (#166) by @harshavardhana in #20601

The bugfix of the heal algorithm is not mentioned in the release note, but I verified this version myself and it is getting better but there still is room for improvements:

- If less than parity shards on the hot tier were lost, the metadata can be healed
- If more than parity shards on the hot tier were lost, the object is lost.
- even if the data is still present in the cold tier, MinIO is not able to recover it from another tier.
For example, I was not able to heal the file 'testfile_241028_idx20' (the only file with size 200KiB) from the first example even though it is still present in the cold tier (right side of the terminal there is a file with size 200KiB).

The bug on inconsistent LIST is still present in RELEASE.2024-11-07T00-52-20Z. The healing fix from two weeks earlier did not address it, so listings can still return results that disagree with the cluster while everything else looks green. Tested directly against the release.

Edit January 9th 2024

No news from MinIO and the bug on inconsistent LIST is still there in RELEASE.2024-12-18T13:15:44Z. Three releases after the reproduction, the listing inconsistency remains the one failure this post keeps re-testing, and it remains unfixed in the releases checked so far.

Alternative object storage solutions with S3 API

If the plan is to leave MinIO, several open source projects keep an S3-compatible API, and they differ in maturity and operational weight. The list below is the one I discuss with clients choosing a replacement for a production cluster.

- [ceph](#) radosGW -> prod ready and battle tested
- [apache ozone](#) -> S3 gateway not mature (as of 1.4.0). I do not recommend except if your clients are using Hadoop API (Hadoop, Spark, etc.)
- [garagehq](#) -> young project but AGPL license
- [SeaweedFS](#) -> young project but my personal favorite
- [RustFS](#) -> not production ready, but very promising and Apache licence

Originally published on [dev.to](#)

[book a call](#) !

Update 2026-09-29. This post is now part of [Object Storage for Data Platforms at Scale](#), the field guide to running object storage at scale.

Related posts

- [Object Storage for Data Platforms at Scale](#)
- [MinIO's Community Edition: the end?](#)
- [MinIO goes into maintenance mode](#)
- [The hidden costs of using HDDs in on-premises MinIO deployments](#)
- [MinIO site replication: sync mode will not save your RPO](#)

PDF snapshot of the public article. The online version is authoritative; links remain clickable.