Empowering HPC and AI Workloads with Optimized Lustre Reliability and Performance

Duncan Vogel

He/Him

Zanhua Huang

He/Him

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lustre Recovery

Resilience through chaos

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Objectives of this Talk

Provide insight into the technical details of Lustre recovery

Discuss some challenges of running Lustre as a managed cloud service

Examine a case study of identifying and improving a Lustre recovery issue that arises in the chaos of cloud workflows

Build the foundation for recovery to get even faster and more reliable

aws

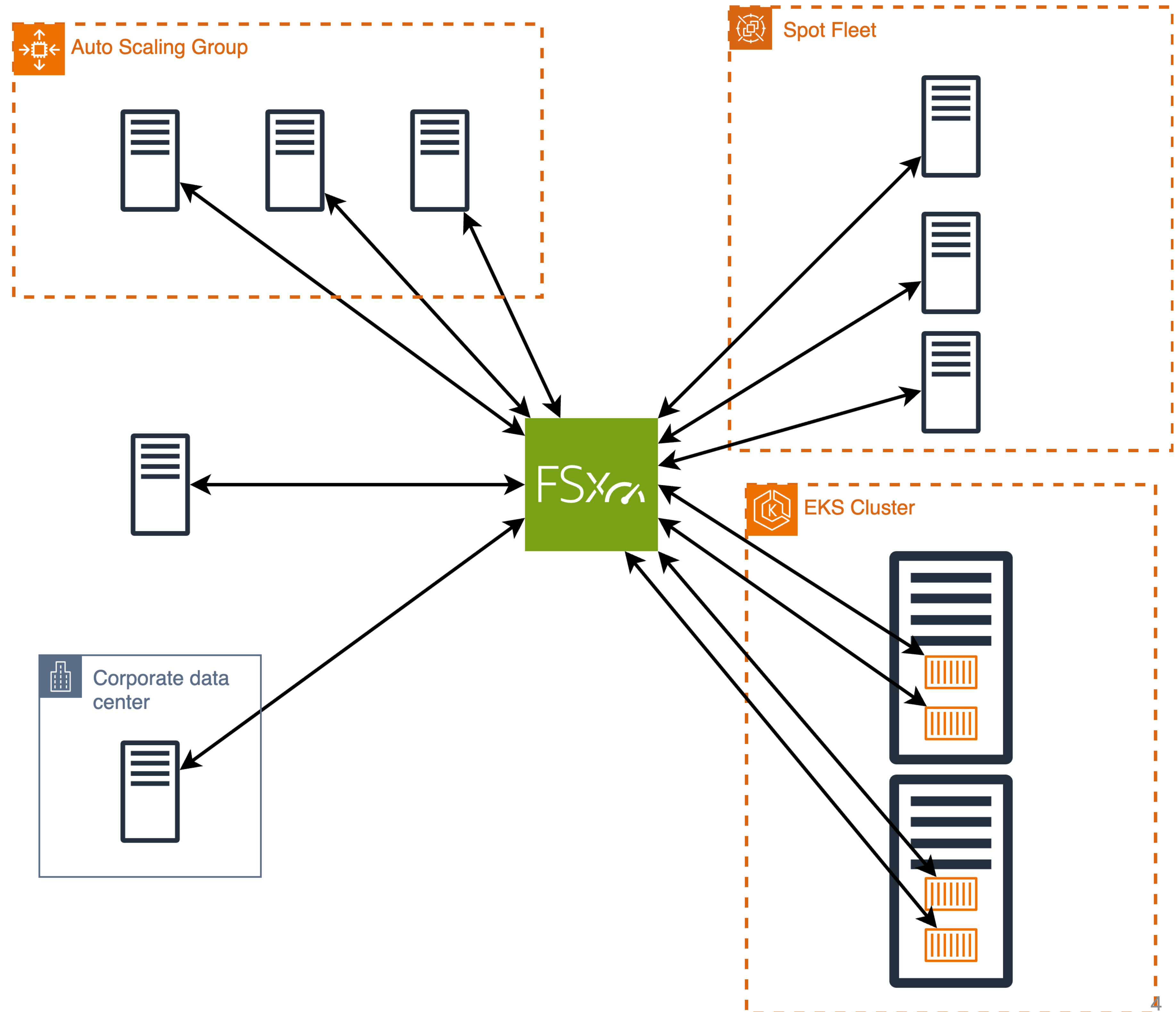
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Power of Cloud Storage

Spin up and down clusters at will using EC2 and EKS

Multi tenancy for customers' teams or their own customers

Access the same underlying storage on hosts of all architectures and sizes



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Challenge of Building Cloud Storage

Spin up and down clusters at will using EC2 and EKS

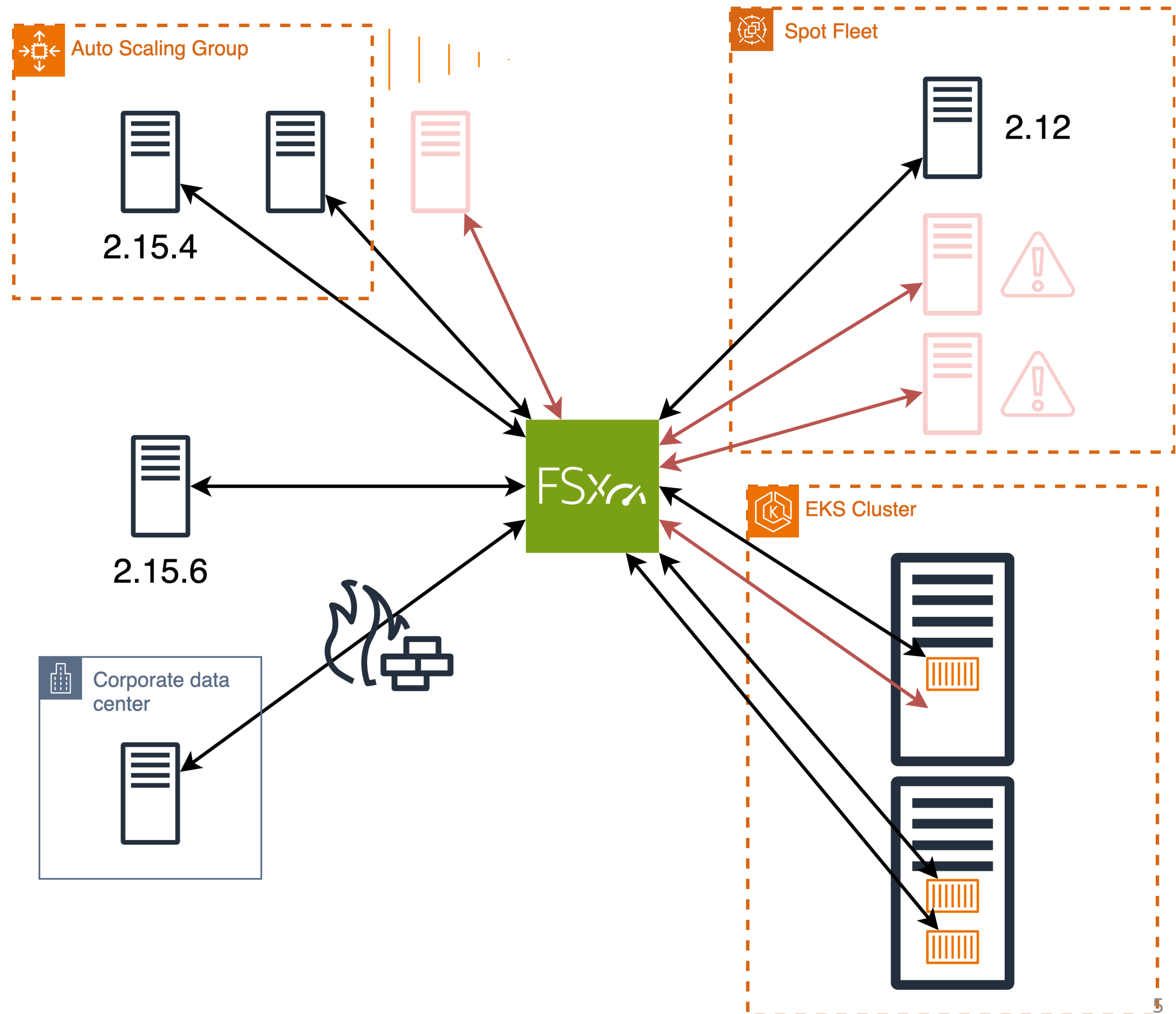
(clients appear and disappear abruptly)

Multi tenancy for customers' teams or their own customers

(variety of client versions, even on the same filesystem)

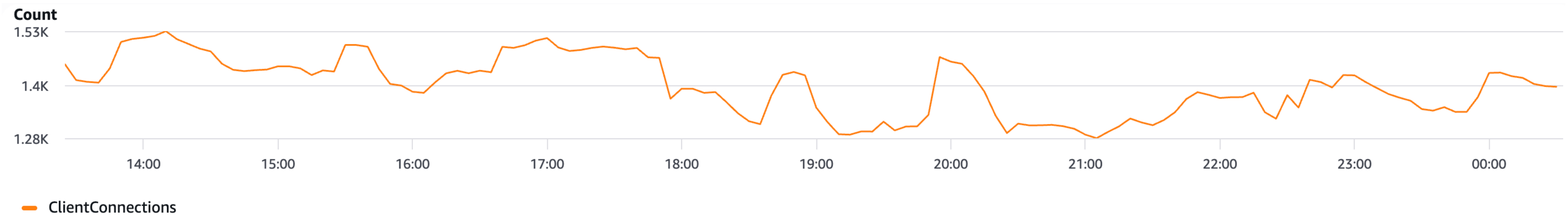
Access the same underlying storage on hosts of all architectures and sizes

(client faults must not affect other healthy clients)



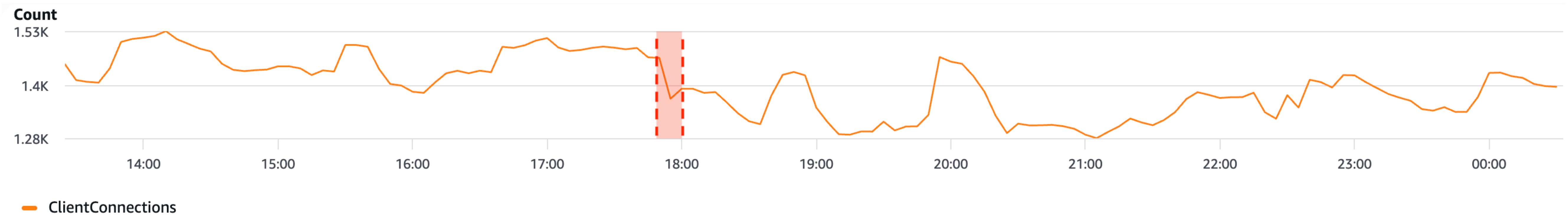
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Challenge of Building Cloud Storage



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Challenge of Building Cloud Storage



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lustre Recovery Overview

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

When do you need it?

Failover

Hardware failure

Security patching

Upgrades

Lustre bug

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What does it involve?

A server target (MDT, OST) rebuilds its in-memory state with the help of its clients

Rebuilds exports (client connections)

Completes RPCs that were being processed before recovery

Re-grants existing LDLM locks

(DNE MDT only) completes any in-progress distributed transactions

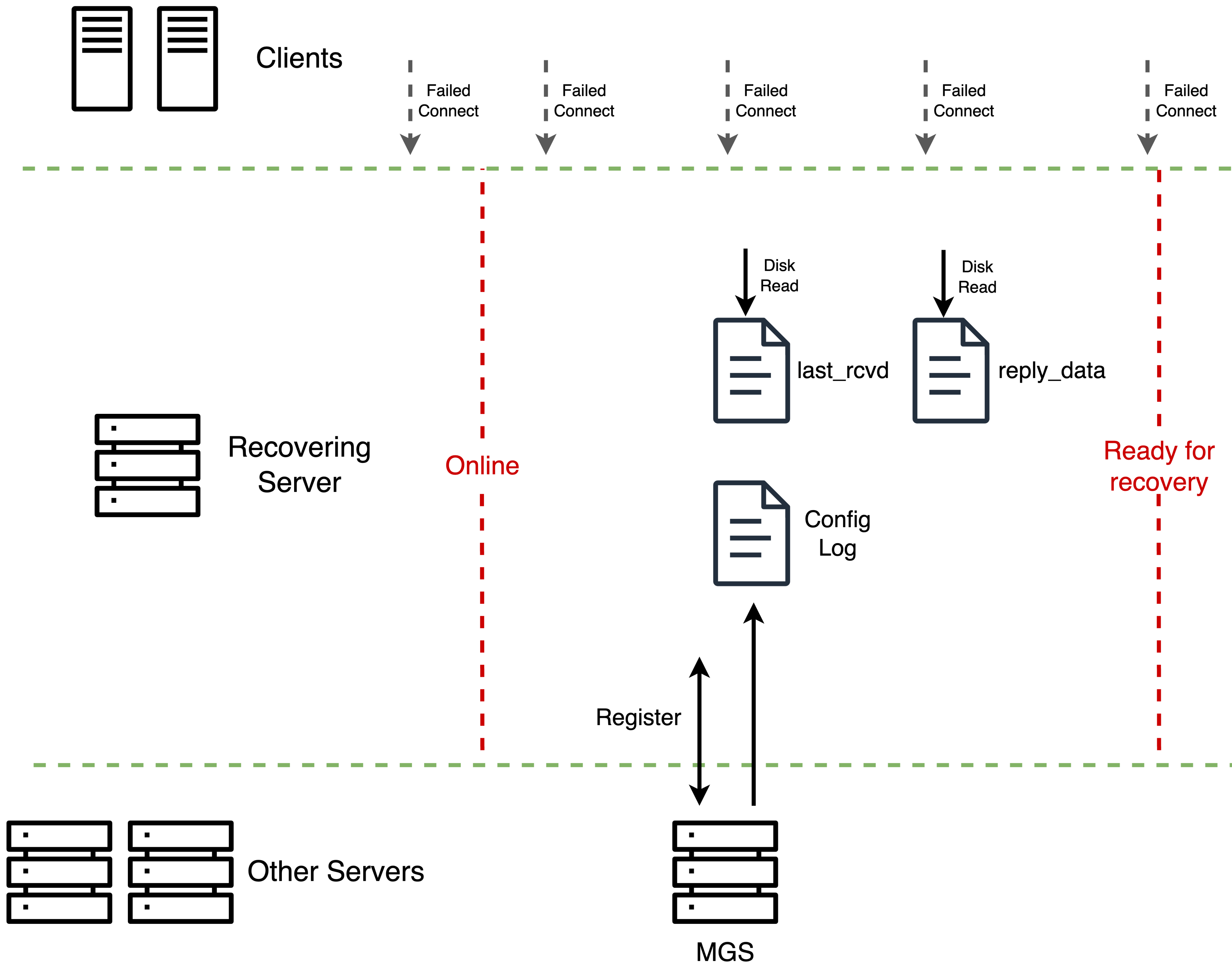
aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Phase 0: Server Synchronization

- Performs normal target initialization
  - Register with and read config log from MGS
  - Start LWP
- Reads last_rcvd and reply_data

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.


Clients
Failed Connect
Failed Connect
Failed Connect
Failed Connect
Failed Connect
Disk Read
Disk Read
last_rcvd
reply_data
Recovering Server
Online
Ready for recovery
Config Log
Register
Other Servers
MGS

Phase 1: Client Reconnection

Depending on the state of the MGS, clients either:

- Are proactively informed by the MGS that the server is ready for recovery
- Continuously try to re-establish connection with backoff

Any requests that were already committed are replayed immediately (for example, opens on open file descriptors or transactions committed but not replied before the server went offline)

Any unused, cached LDLM locks are dropped

Any in-flight RPCs that were not in progress will wait for recovery to complete

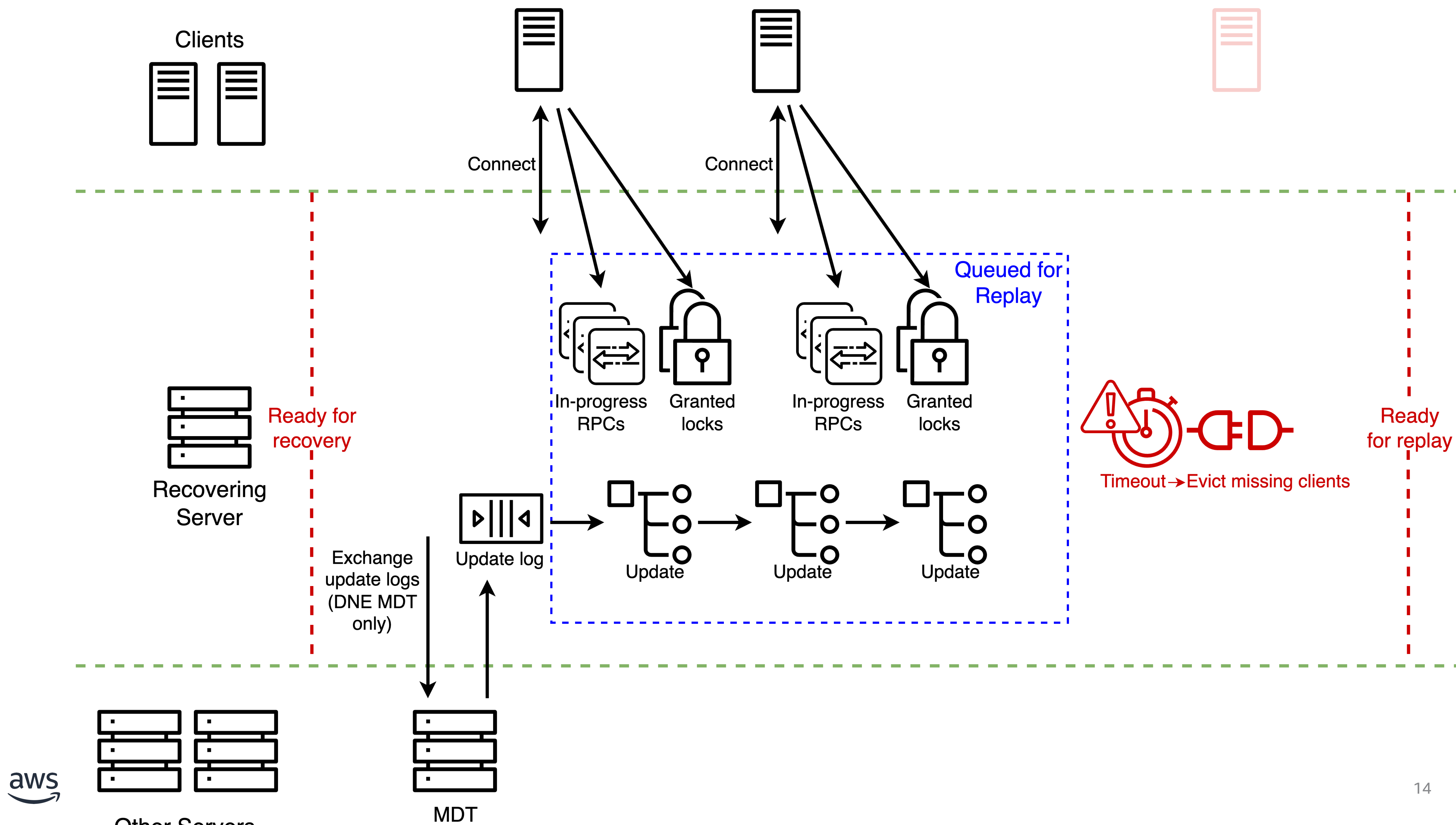
Any incomplete, in-progress RPCs and in-use LDLM locks are added to the replay queue in transaction number order

Complete once every client has reconnected and finished queueing replays (or recovery times out)

On DNE MDTs, this phase also has the responsibility of syncing the remote update_log llog records, which catalog the in-progress cross-MDT operations

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Clients
Connect
Connect
Queued for Replay
In-progress RPCs
Granted locks
In-progress RPCs
Granted locks
Recovering Server
Ready for recovery
Timeout→Evict missing clients
Ready for replay
Update log
Update
Update
Update
Exchange update logs (DNE MDT only)
Other Servers
MDT
aws

Phase 2: Replay Requests and Locks

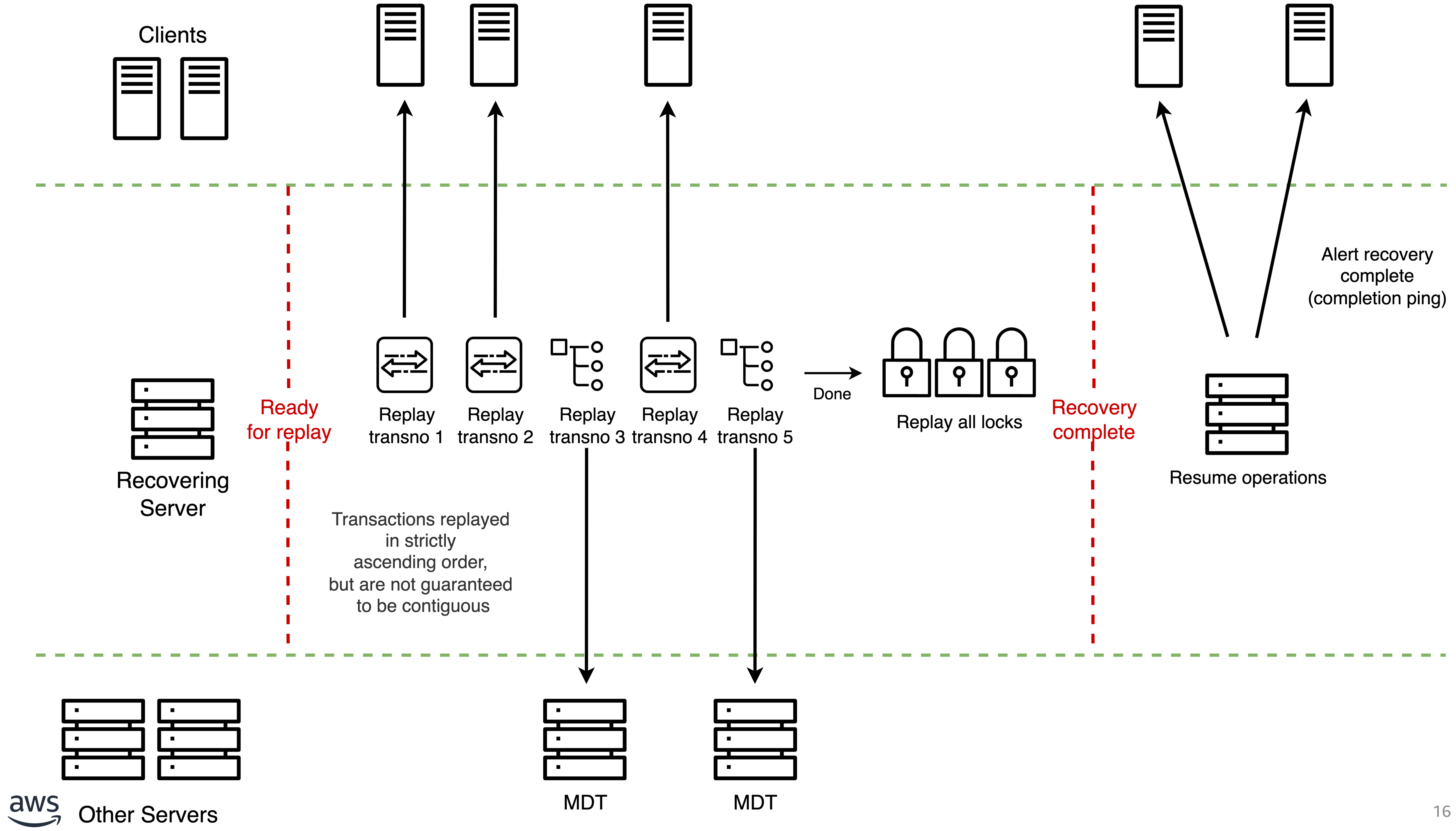
Replay all requests in transaction number order

Replay all locks in transaction number order

If all clients joined recovery, the server has a complete picture, so can just replay everything in order

If recovery timed out and clients were evicted, there may be gaps in the replay queue, so a feature called Version Based Recovery (VBR) is used. This feature compares the object version each request operated on before to the current version, and rejects updates where there is a version mismatch

aws © 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.


Clients
Ready for replay
Replay transno 1
Replay transno 2
Replay transno 3
Replay transno 4
Replay transno 5
Done
Replay all locks
Recovery complete
Alert recovery complete (completion ping)
Resume operations
Recovering Server
Transactions replayed in strictly ascending order, but are not guaranteed to be contiguous
MDT
MDT
aws
Other Servers

The Unhappy Path

Soft timeout: obd_timeout * 3 (5 minutes)

Hard timeout: obd_timeout * 9 (15 minutes)

Soft timeout controls how long Lustre will wait for missing clients to join recovery

When soft timeout is reached, the server will **evict** all clients which have not joined recovery and enable VBR

Typically, all the clients that joined recovery before the timeout can successfully replay using VBR and resume operations

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The REALLY Unhappy Path

If the hard timeout is reached, Lustre decides it's had enough of replay and needs to get back to work

Every client that has not completely finished recovery is evicted

If we never enabled VBR, the only clients which survive were clients whose transactions were committed before the server went offline but just didn't hear back

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Adaptive Timeouts

The RPC timeout will be dynamically updated to account for observed RPC round trip latency

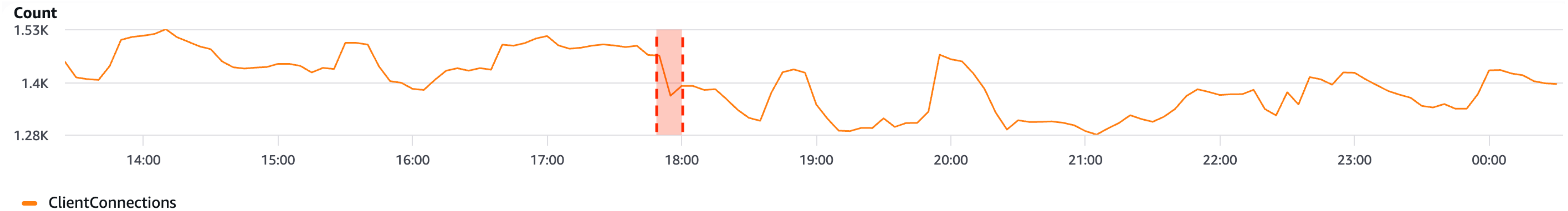
The server accounts for an additional value called service_timeout which records the processing time not accounting for network latency

During recovery, the server will read the service_timeout from reconnecting clients and accept it as a baseline service_timeout, and extend the recovery timer to be at least service_timeout, plus 25%, plus 125 seconds (accounting for many things that might contribute to latency, plus an “arbitrary minimum”)

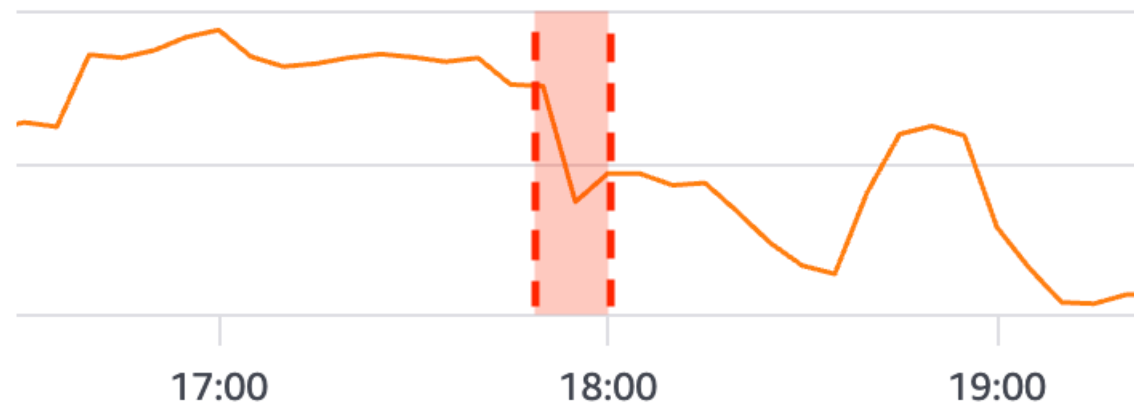
aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Case Study: Hardware Failure during Ephemeral Workload



Hardware failure event window



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Event

A high intensity workload running on EC2 Spot instances is using the filesystem for a variety of jobs, and new clients are mounting and disconnecting as jobs run

Hardware problems lead to host degradation on the single MDS, slowing operations to a crawl

Adaptive timeouts rise due to stalled requests on the already overloaded filesystem, all the way to at_max of 600 seconds

AWS monitoring detects the failure and migrates the MDS to new hardware

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Recovery

While the MDS is being migrated to healthy hardware, ephemeral instances are terminated

The MDS comes back online on the new hardware and the MDT enters recovery

The online clients rejoin recovery quickly, and share their adaptive timeout value of 600 seconds

The MDT extends recovery timeout to (600*1.25)+125 = 875 seconds

Because of clients which disconnected while the MDT was offline, recovery waits the full 875 seconds for these clients before timing out

25 seconds is not long enough to finish replays

When recovery reaches the hard timeout of 900 seconds, recovery aborts, and nearly all the remaining clients are evicted

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Problem

Clients disconnecting while a server is offline will always result in a timeout

The server will accept adaptive timeouts from clients, even if the timeout is based on conditions which are no longer true

If clients are missing, we **need** to evict them and enter VBR to recover the remaining clients

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Fix

LU-20135

Don’t allow clients to extend the recovery timeout on reconnect

When using failover via Pacemaker etc., the problem becomes more complicated, but the fundamental issue remains the same

Moral – every timeout in Lustre is part of a hierarchy, and incorrectly estimating a timeout at a lower layer can end up encountering different timeouts in higher layers

Result: 88% fewer hard timeouts

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

The Future

LU-19199

Simplify more of the recovery code

Faster reconnection and faster replay

Further optimize recovery for chaos

aws

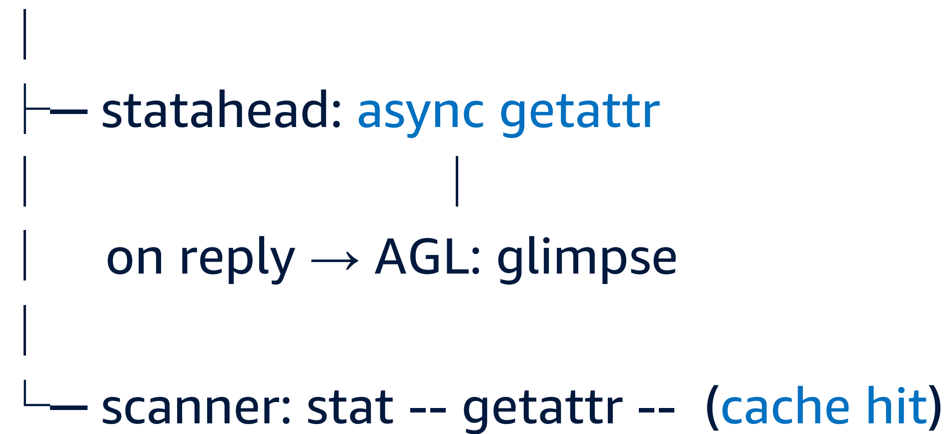
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Async xattr Statahead

Closing the ls –l Gap in Lustre

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is Statahead?



Same num of RPCs, but ahead of time

Companion thread: AGL (async glimpse lock) prefetches file size from OSTs

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Where Statahead Falls Short

ls -l doesn't just need stat. Modern distros also need:

- SELinux label (security.selinux)
- POSIX ACL (system.posix_acl_access)

xattrs are not pre-fetched

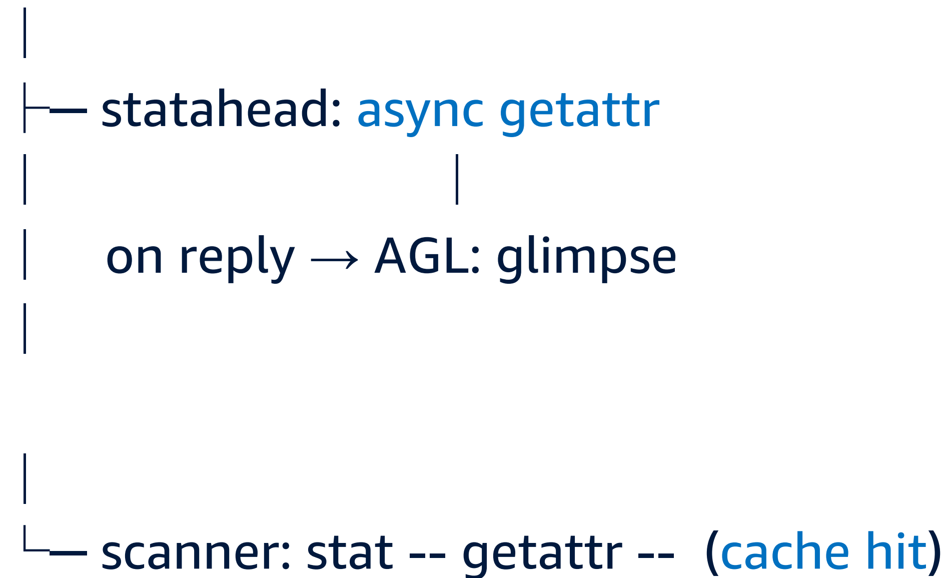
ls –l scanner still pays synchronous xattr RPCs per file after the cached stat hit.

Configuration	10K Files ls -l
No statahead	~12 s
Statahead (stat only)	~11 s

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Our Change: Async xattr Statahead

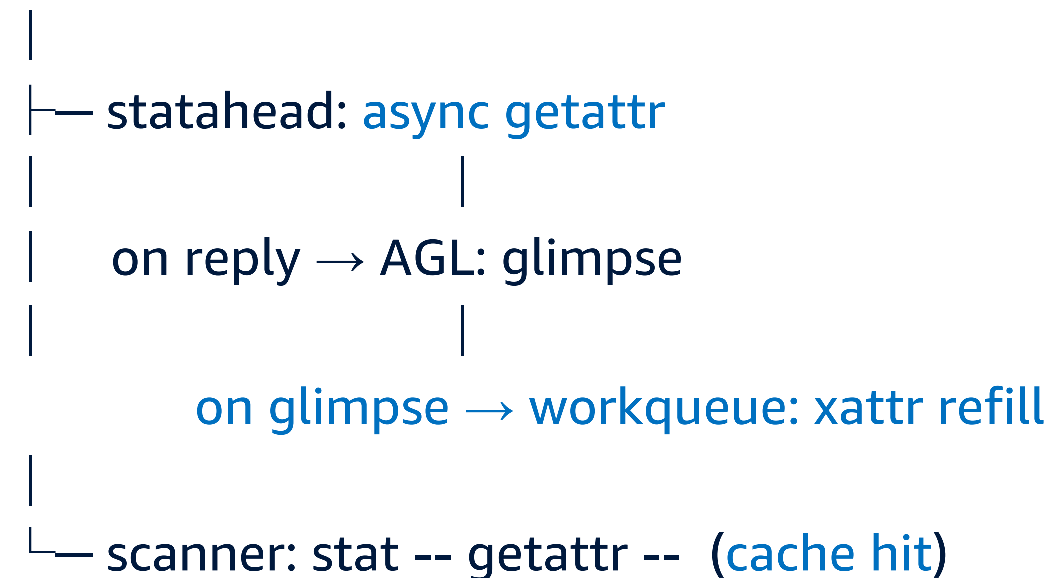
Piggyback on the existing AGL pipeline.



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Our Change: Async xattr Statahead

Piggyback on the existing AGL pipeline. New knob: statahead_xattr (default off).



After each AGL completes, schedule a work_struct that calls ll_xattr_cache_refill() — one RPC pulls all xattrs.

LU-17239

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

Folder	Operation	stat+xattr statahead	stat only statahead	no statahead
10K	ls -l	2.2 s	11.4 s	12.3 s
100K	ls -l	25 s	1m49s	4m03s
1M	ls -l	4m04s	18m33s	41m07s
1M nested	find	17m44s	32m29s	1h13m

Lustre 2.15.6, AL2023 client, statahead_max=512
10K/100K/1M = flat directory with that many files. 1M nested = 1M files across a hierarchy of subdirectories.

5 - 20× faster on ls -l across AL2023, RHEL9, Ubuntu 22/24

du on flat dirs:

- gnulib's fts.c sorts entries by inode for large dirs, which scrambles Lustre's access pattern so statahead never activates
- Fixed upstream in gnulib 578b8d7

lfs find doesn't benefit:

- Uses its own LL_IOC_MDC_GETINFO ioctl that bundles stat + layout in one RPC, bypassing the VFS stat path where statahead lives

aws

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

On-going / Future Work

Cover non-regular files

xattr prefetch today only fires for regular files

Merge LU-17239

patch in progress, needs more work (add tests, address feedback, etc)

aws
© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Summary

- Current statahead prefetches sattr but not xattr
- statahead_xattr closes that gap
  - 5–20× speedups on ls -l, find, ls -lR across distros
- Next:
  - Extend to non-regular files,
  - Merge in LU-17239

aws © 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Duncan Vogel

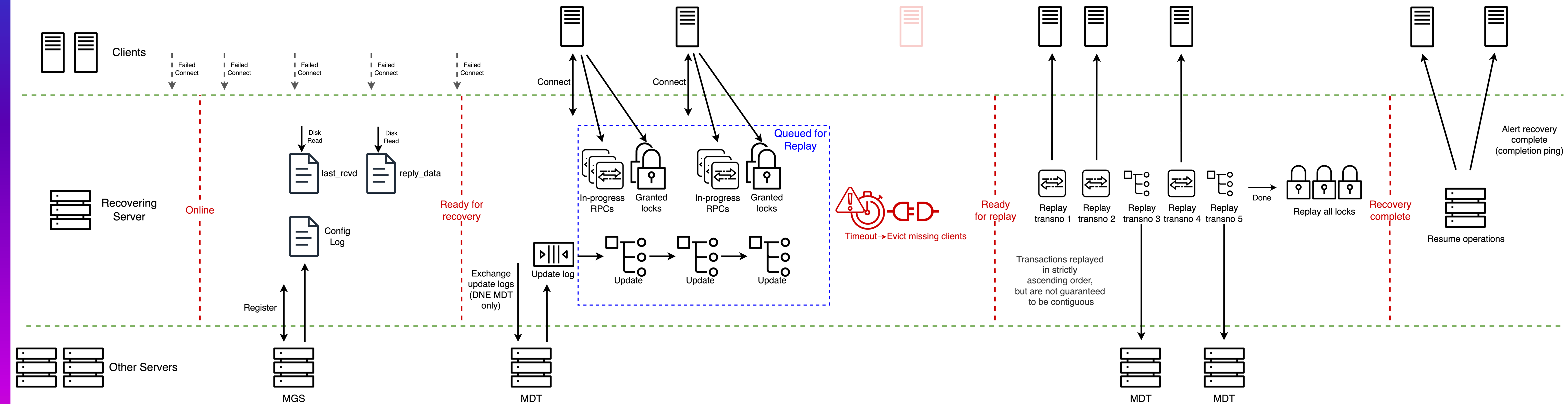
fvogdunc@amazon.com

Zanhua Huang

zanhua@amazon.com

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Appendix: Full Recovery Diagram



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.