OAK RIDGE National Laboratory

Efficient Metadata Scanning Using ZFS and NVMe-over-Fabrics

Ross G. Miller

rgmiller@ornl.gov

LUG ‘21

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

Introduction

First things first: THIS IS A PROTOTYPE!

- We haven't implemented this in production (and probably won't)

Why do this?

- Want to know what's happening on our filesystem
- Traditional techniques don't scale well
  - Running `find` from a client can take hours (or more likely, days) and can heavily load the MDS
  - `lfs find` is better but still not fast enough
  - Lustre changelog can't keep up

Basic Design Goals

- Efficiently & quickly scan the filesystem
  - In this case, “quickly” means “an hour or so”
  - Actually, anything faster than one day is acceptable
- Don’t overload the MDS
  - We don’t want the scanning operation to noticeably slow down normal filesystem operations
- Incremental scans are preferred, but we do need to be able to do a full rescan if necessary.
  - Incremental scans are probably necessary to meet the two previous requirements

ZFS Snapshots?

ZFS snapshots are potentially very useful:

- Low cost to create & destroy (nearly instant)
- Keeping them around only uses space for files that have changed
- They allow us to have a consistent view into the filesystem
- `zfs diff` can report differences between two snapshots

This all sounds promising, BUT…

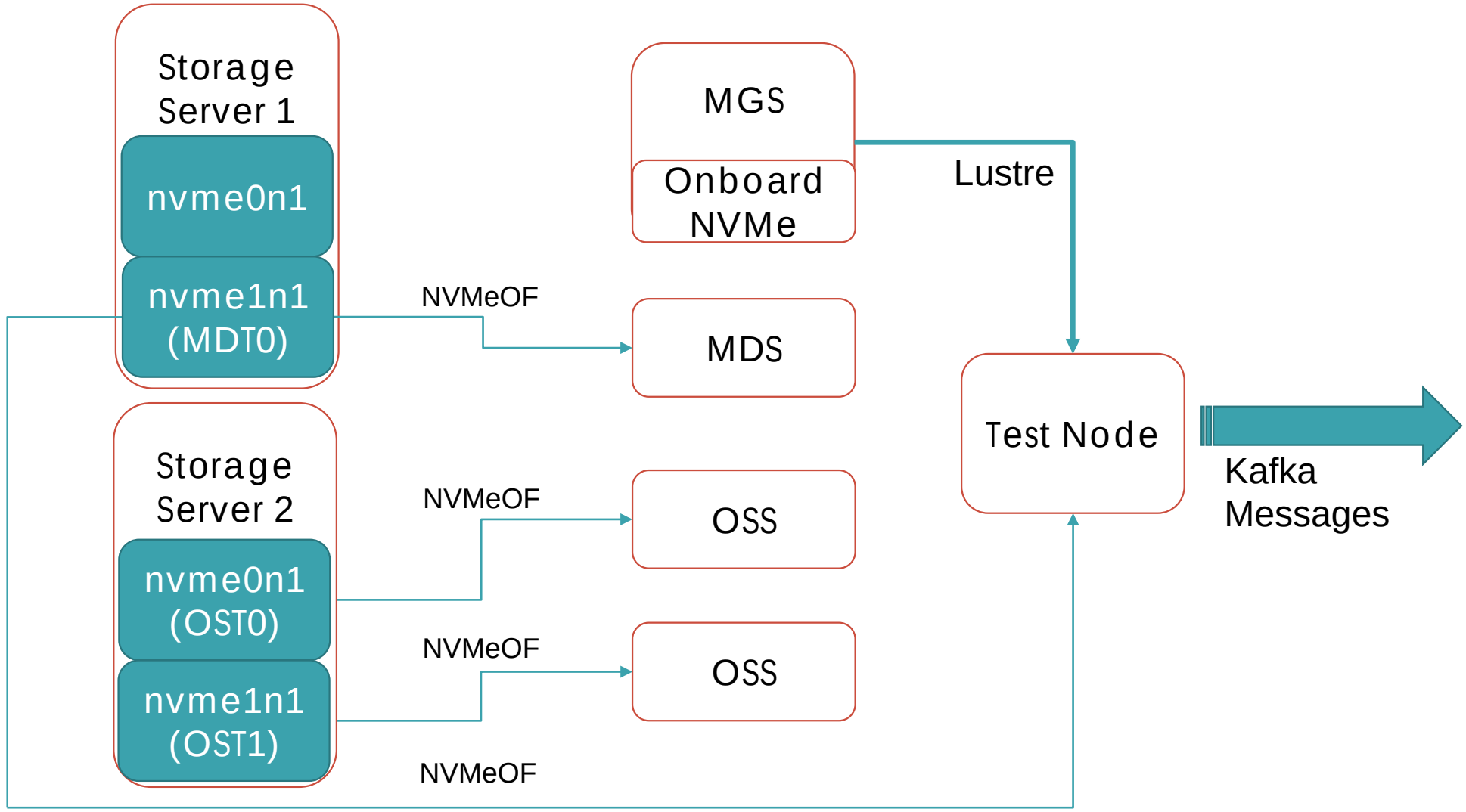
…"Normal" ZFS diff won't work

- In order to actually run `zfs diff`, the ZFS filesystem has to be mounted
- For performance reasons, Lustre doesn't actually mount the ZFS filesystem(s) that are the backing-store for OSTs & MDTs.

How to resolve this impasse?

Use NVMe-Over-Fabrics to DUAL MOUNT the zpool for the MDT(!)

Test Hardware



Order of Operations

1. Create a new snapshot - lctl snapshot_create
2. Import the zpool read-only
3. Mount filesystem (again, read-only)
4. Mount the Lustre snapshot (also read-only)
5. Run zfs diff on the 2 most recent snapshots
    a. Pipe the diff output into a parser script that will stat the files, do any pre-processing and push the results out to Kafka & Elasticsearch.
6. Unmount Lustre & ZFS filesystems and export the zpool
7. (Optional) Delete old snapshot

Ensuring No Writes To The ZPool

- `zpool import` has a read-only flag
- Checked with the developers and confirmed by inspecting the ZFS source code: when the read-only flag is used, the zdev devices are all opened read-only

- Considered using cgroups to add an additional level of safety
  - Didn't get a chance to actually try this out though

Performance Numbers

(For test setup using a single NVMe device for the MDT)

- Diff's / second: ~1900 (~100% CPU)
- Stat's / second: ~11K (~98% CPU)
- Kafka message ingest rate: >50K
- Combined (diff+stat+message publication): 1900/s (~135% CPU)

Does this scale up?

`zfs diff` seems to be the limiting factor: 1 thread, 100% CPU.

- That's 6.8M changed files per hour.
- In a filesystem with 5 billion files, that's 0.13%. Is that sufficient?

The Expense Of ‘stat’

- A lot of the expense of the stat() function comes from the need to get size data from every single OSS that the file is striped across.
- For this application, LSOM (Lazy Size On MDT) would be sufficient.
  - Landed in v2.12
- Need some mechanism to tell Lustre to reply to stat requests with lazy size
  - LU-10934 implements this (using the statx() call instead of stat())
  - Landed in v2.14

Incremental Scan vs. Full Scan

Diffing two snapshots is efficient for incremental scans, but what if we have to do a full (re)scan of the filesystem?

Answer: Take a snapshot right after the filesystem is set up and keep it forever. To do a full (re)scan, diff the latest snapshot against this baseline snapshot.

- Since the filesystem is empty when the base snapshot is created, there's essentially zero cost to keeping this snapshot.

Thoughts & Conclusions

- This design only works for a fairly niche use-case:
  - MDT's must use ZFS and NVMe devices
  - Networking that supports NVMe-over-Fabrics
  - A system large enough that other methods don't work
  - Normal workloads that aren't limited by IOPS
- Actual operations would be perilous
  - One simple mistake – forgetting the read-only flag on the import – could result in your MDT being destroyed
  - It was also unclear if we could ever *prove* that this technique is safe
    - Best we could say was "It didn't corrupt the MDT during our testing."

Closing Remarks & Questions

Acknowledgement:
This research used the resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is managed by UT Battelle, LLC for the U.S. Department of Energy, under the contract No. DEAC05-00OR22725.