Balancing storage utilization across a global namespace

Manish Motwani
Cleversafe, Inc.
Agenda

- Introduction
- What are namespaces, why we need them
- Compare different types of namespaces
- Why we need to rebalance data
- Rebalancing scenarios, requirements, design
- Our observations
- Future work
- Conclusion
Details not covered

- Data format on disk
- Network protocols
- Namespace persistence
- Security
Introduction
Name lookup

Requirements:
- Scalable
- Highly available
- Highly reliable
- High performance
- Fault tolerant

Some solutions:
- HA clusters
- More memory
- Synchronization
- …
Achieving unlimited scale

- Static name lookup
- Large enough namespace to store practically unlimited number of objects
- Uniform HW utilization
- Independent scaling of storage capacity and performance
- Hardware virtualization
  - Tolerate disk/node failures
  - Replace disks/nodes with newer, larger ones
Namespaces
Namespaces

- A namespace is a set of all possible names that can uniquely identify objects.
- Used by clients and servers to determine where each piece of data is written to and read from.
- Hierarchical namespaces
  - E.g., file systems.
- Flat namespaces
  - Dense namespace (e.g., a block device)
  - Sparse namespace
    - Infinite (integers, whole numbers, prime numbers, etc.)
    - Finite (a range of an infinite namespace, for e.g., 0-2^{500})
Contiguous vs. fragmented

- Fragmented (non-contiguous) namespaces require central metadata management for name-mapping.
- At petabyte scale, central metadata management is extremely difficult. At exabyte scale it’s impossible.
A global namespace is a very sparse namespace such that we can create billions of random names in it every day and still never see a conflict or run out of names.

Cleversafe namespace

- \(2^{384}\) names, broken into storage and routable names
- Routable name: Vault identifier, storage sub type, generation id, slice index
- Storage name: \(2^{192}\) names (range: 0 – \(2^{192}-1\)
  - Generally expressed in hex
  - Name is stored in 24 bytes
Namespace mapping

- Global namespace
- Vault
  - Site 1
  - Site 2
  - Site 3
- Node 1
  - Disk 1
- Node 2
  - Disk 2
- Node 3
  - Disk 3
  - Disk 4
- Node n - 1
  - Disk 5
  - Disk 6
  - Disk 7
  - Disk 8
- Node n
  - Disk 9
  - Disk 10

Memory addresses:
- 0000...00
- FFFF...FF
- 000...1FF
- 200...3FF
- 400...5FF
- C00...DFF
- E00...FFF
- 40...
- 5FF...
Data Rebalancing
What is data rebalancing?

- Rebalancing is the changing of the entity responsible for storing some portion of the namespace.
  - Usually requires data migration.
  - Performed on a live system.
  - Done to achieve more uniform utilization for higher write availability.
  - E.g., changing ownership of namespace range from one disk to another, or one storage node to another.
Why do we need to rebalance data?

- Hardware failures
- Disk/node removal
- Disk/node addition/replacement
- Non-random name generation
- Data needs to live forever – HW doesn’t
Disk usage example

- 80% in the 1-5 range
- 40% in the 6-10 range
Simple rebalancing

80% 40% 80% 60% 60%

1-5 6-10 1-3 4-10
Rebalancing components

- Rebalance policy
  - Look at global view to determine general direction
  - Automatically decide on source and destination based on urgency

- Data movement
  - Move data from source to destination
  - Guarantee internal consistency
Rebalance Policy
Rebalance policy

- **Goal**
  - Writes should not fail due to full disks if there is space on other disks
  - Optimize on least amount of data movements
  - Do only when necessary
  - Fully automated – 0 administrative supervision

- **Design considerations**
  - Ongoing writes keep changing the picture
  - Different sized disks
  - Disk failures
  - Data rebuilding
  - Impact on system performance
Usage example

Ascending name order
Usage example

Ascending name order
Usage example

Ascending name order
Usage example

Ascending name order

Average fill level
Rebalance policy

Algorithm

- Determine highest utilized disk (% usage)
- Determine direction of data movement
  - From the greater of average utilization left of the disk or right of the disk
- Move small chunk of data (~1GB)
- Repeat until an imbalance ratio is below a pre-configured threshold
Simulation 1 – Disk replacement

Report interval: 1 Hour
Write rate 80,000 MB/s
Rebalance rate 80,000 MB/s
Each disk size: 3 TB
Simulation 2

Report interval: 4 Hours      Write rate 5 MB/s      Rebalance rate 80 MB/s      3TB disks
Making smarter policy decisions

- Some of the problems encountered
  - Smaller/empty vaults move too much range
  - Disk utilization is not always indicative
    - Rebuild hasn’t occurred after
      - Outage
      - Disk replacement
    - SS is running slow so client is dropping writes
  - Too much load on the system

- Some solutions
  - Move highest utilized vaults first
  - Take remote usage into account
  - Limit number of rebalancing threads in our scheduler
Rebalancing Design
Implementation requirements

- All data "available" during rebalancing
  - No operations should be disrupted, however performance might be affected
  - Work with existing storage interface
    - Independent of underlying storage mechanism

- Maintain namespace contiguity
  - Rebalancing performance is not critical – it is usually a background process
  - No additional metadata should be added to namespace mapping (but existing can change)
Storage interface

- Read(name) – read all revisions of name from media
- Write(name, revision, txid, data) – write a transactional piece of data to media
- Commit(txid) – commit a transaction
- Rollback(txid) – rollback a transaction
- List(range)
Data move design

- Copy, Sync, Delete "chunks"
- Asynchronous and lockless – can also be used for inter-node rebalancing
- Source store (local store) is the main actor
- Policy decides range to move
- Non-interruptible- entire range is moved; if failure occurs, it’s rolled back
Data move design (cont.)

--- No migration

- Local master
  - copy data
    - a) copy uncommitted
    - b) copy committed
  - writes to both remote & local
  - reads from local

- Remote master
  - writes only to remote (proxy)
  - reads from remote

1. Transition to local master
   - snapshot of all open tx created
   - "sync" point for uncommitted data
   - pause new commits until inflight commits are completed

2. Transition to remote master
   - "sync" point for both committed & uncommitted data

3. Transition to no migration
Future work

- Inter-node rebalancing
- Temporary node namespace fragmentation
Conclusions

- Global contiguous namespaces combined with no central metadata management can allow storage systems to scale to exabytes and beyond.
- Data rebalancing is required to compensate for media failures or in heterogeneous environments.
- Automated rebalancing decision algorithms allow for minimal administrative supervision.
- Data movement is faster and more scalable when not dependent on client transactions.
- There’s always room for improvement.
Questions?