Ceph: A Scalable, High-Performance Distributed File System

S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long
Presented by Philip Snowberger

Department of Computer Science and Engineering
University of Notre Dame

April 20, 2007

Outline

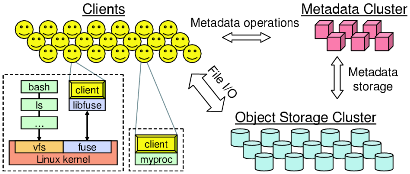
Problem

- Distributed filesystems allow aggregation of resources
  - Can increase fault-tolerance
  - Can increase performance
  - Increases complexity
- Metadata is a bottleneck in many distributed filesystems
  - Centralized metadata
  - Distributed metadata
- Can we make a distributed filesystem that scales on both data and metadata operations?

Goals of Ceph

- Achieve scalability to petabyte workloads, while maintaining
  - Performance
  - Reliability
- Scalability?
  - Storage capacity
  - Throughput
  - Scale the above while maintaining useful performance for individuals

How Does Ceph Attempt to Accomplish This?



- Decoupling data and metadata
  - A Ceph cluster consists of servers responsible for storing objects, and servers responsible for managing metadata
- *Dynamic* distributed metadata management
  - Robust against failures and workload changes
- Reliable Autonomic Distributed Object Storage
  - Leverage “intelligence” available at each node in a cluster

What are Metadata?

- Metadata are information about data
  - Length, permissions, creator, modification time, . . .
  - File name?
- Almost every filesystem access affects metadata
- Different types of metadata can have different consistency requirements

Traditional Block Storage

- In traditional block storage, one piece of metadata is the allocation list
  - Sequence of disk block ranges that comprise the data of the file
- Managing this list takes a significant amount of computrons
- In a distributed setting, disk blocks are too low-level an abstraction

Object Storage

- *Objects* consist of paired data and metadata
- An Object storage device is responsible for keeping track of where's the object's bytes are on disk
- Thus, object storage relies on intelligence at storage nodes to relieve some of the management load

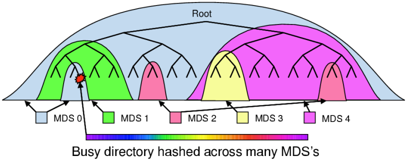
A Simple Object Storage System

- Consider a simple distributed filesystem
  - Centralized directory server: "Where is /tmp/foo?"
  - Distributed file servers: "Give me bytes 9043-43880 of /tmp/foo"
- This design does not scale:
  - Can the directory server handle 10,000 requests/second? 1,000,000?
  - Can a single file server serve up a 10 MB file to 1,000 hosts?

Metadata Distribution in Ceph

- In the past, distributed filesystems have used static sub-tree partitioning to distribute filesystem load
  - This does not perform optimally for some cases (e.g. `/tmp`, `/var/run/log`)
  - It also performs poorly when the workload changes
- To offset this lack of optimality, more recent distributed filesystems have opted to distribute metadata with a hashing function
  - This removes the performance gains from directory locality
- Ceph uses a new approach, dynamic sub-tree partitioning

Dynamic Sub-Tree Partitioning

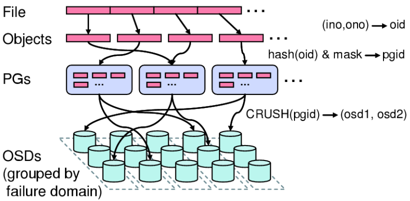


- Each MDS is responsible for some sub-tree of the filesystem hierarchy
- Whenever an operation "visits" an inode (directory or file), the MDS increments that inode's time-decay counter
- MDSs compare their counter values periodically
- When an imbalance is detected, the MDS cluster reassigns the responsiblity over some sub-trees to balance the counter values
- Extremely busy directories can be hashed across multiple MDSs

A Naïve Object Placement Method

- To find where to put a chunk of a file, $hash(inode.chunkNum) mod NumServers$
- But what happens when a server goes down or we add a server?
  - The hashing function needs a new modulus
- In a “petascale” distributed filesystem, failures and expansion must be regarded as the rule, rather than the exception
- Is there a better way to place chunks?

Distributing Objects in Ceph



- Each object is mapped to a Placement Group by a simple hash function with an adjustable bitmask
  - This bitmask controls the number of PGs
- Placement Groups (PGs) are mapped to each Object Storage Device (OSD) by a special mapping function, CRUSH
  - Number of PGs per OSD affects load balancing

CRUSH

- A special-purpose mapping function
- Given as input a PG identifier, a cluster map, and a set of placement rules, it deterministically maps each PG to a sequence of OSDs, $\vec{R}$.
- This sequence is pseudo-random but deteministic
- "With high probability", this achieves good distribution of objects and metadata
- This distribution is called "declustered"

Cluster Maps

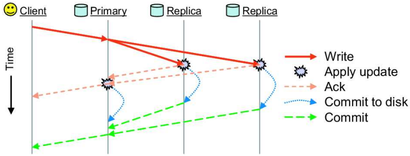
- A cluster map is composed of devices and buckets
  - **Devices** are leaf nodes
  - **Buckets** may contain devices or other buckets
- The OSDs that make up a cluster or group of clusters can be organized into a hierarchy
- This hierarchy can reflect the physical or logical layout of the cluster or network
  - Room123 (*root*)
  - Row1, Row2, . . . , Row8
  - Cabinet1, Cabinet2, . . . , Cabinet16
  - Disk1, Disk2, . . . , Disk256

Placement Rules

- Specify how the replicas of a PG should be placed on the cluster
- The following example distributes three replicas across single disks in each of three cabinets, all in the same row
- This pattern reduces or eliminates inter-row replication traffic

Action	Resulting $\vec{i}$
take(Room123)	Room123
select(1,row)	Row2
select(3,cabinet)	Cabinet4 Cabinet8 Cabinet9
select(1,disk)	Disk44 Disk509 Disk612

Safety vs. Synchronization



- When a client writes data, it sends the write to the Primary
- The Primary forwards the data to the Secondaries, who *ack* that they've received the data and it's been applied to their page caches
- When the Primary receives all the Secondary *ack*s, it returns an *ack* back to the client
- At this point, the client knows that any other client accessing the object will see a consistent view of it

Safety vs. Synchronization (continued)

- So how does Ceph treat data safety?
- Each OSD aggressively flushes its caches to secondary storage
- When the Secondaries have flushed each update, they send a *commit* message to the Primary
- After collecting all the Secondaries' *commit*s, the Primary sends a *commit* to the client
- Clients keep their updates buffered until receipt of a *commit* message from the Primary
  - (Why is this necessary?)

Commonality of Failures

- Failures must be assumed in a "petascale" filesystem consisting of hundreds or thousands of disks
- Centralized failure monitoring:
  - Places a lot of load on the network
  - Can not see "through" a network partition
- Can we distribute monitoring of failures?

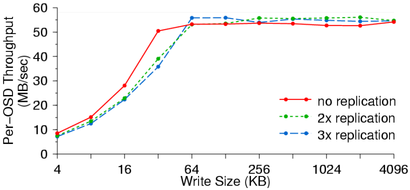
Monitors

- Monitors are processes that keep track of transient and systemic failures in the cluster
- They are responsible with providing access to a consistent cluster map
- When the monitors change the cluster map, they propagate that change to the affected OSDs
  - The updated cluster map (since it is small) propagates via other inter-OSD communication to the whole cluster
- When an OSD receives an updated map, it determines if the ownership of any of its PGs have changed
  - If so, it directly connects to the other OSD and replicates its PG there

So What If A Rack Of Servers Explodes?

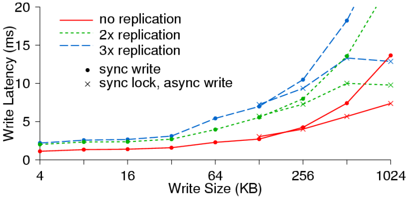
- Each OSD keeps track of the last time it heard from each other server it shares a Placement Group with (replication traffic serves as heartbeats)
- When a node goes down, it isn't heard from in a short time, and is marked *down* (but not *out*) by the monitors.
- If the node doesn't recover quickly, it is marked *out*, and another OSD joins each of the PGs that was affected in order to bring the replication level back up
- Replication of data on the down/out node is prioritized by the other OSDs

Throughput and Latency



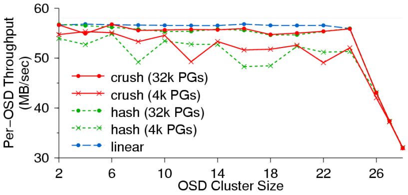
- 14-node OSD cluster
- Load is generated by 400 clients running on 20 other nodes
- Plateau indicates the physical limitation of disk throughput

Throughput and Latency (continued)



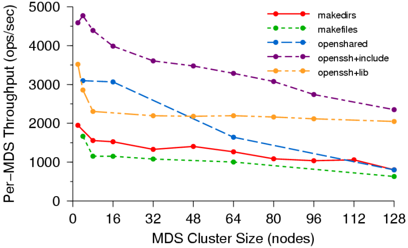
- No difference for low write sizes between two and three replicas
- At higher write sizes, network transmission times dominate network latency

Throughput and Latency (continued)



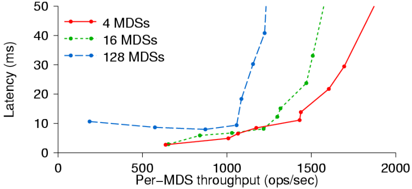
- OSD throughput scales linearly with the size of the OSD cluster until the network switch is saturated
- More PGs even load out more, giving better per-node throughput

Metadata Operation Scaling



- 430-node cluster, varying number of MDSs
- Metadata-only workloads
- Only a per-node throughput slowdown of 50% for large clusters

Metadata Operation Scaling (continued)



- From the *makedirs* workload
- Larger clusters have less-optimal metadata distributions, resulting in lower throughput
- However, this is still very much adequate and performant for a large distributed filesystem

Summary

- Ceph is a distributed filesystem that scales to extremely high loads and storage capacities
- Latency of Ceph operations scales well with the number of nodes in the cluster, the size of reads/writes, and the replication factor
- By offering slightly non-POSIX semantics, they achieve big performance wins for scientific workloads
- Distributing load with a cluster-wide mapping function (CRUSH) is both effective and performant