# Ceph: A Scalable, High-Performance Distributed File System (OSDI '06)

한조(공정훈, 박연홍)
Architecture and Code optimization Lab.
Seoul National University

# Contents

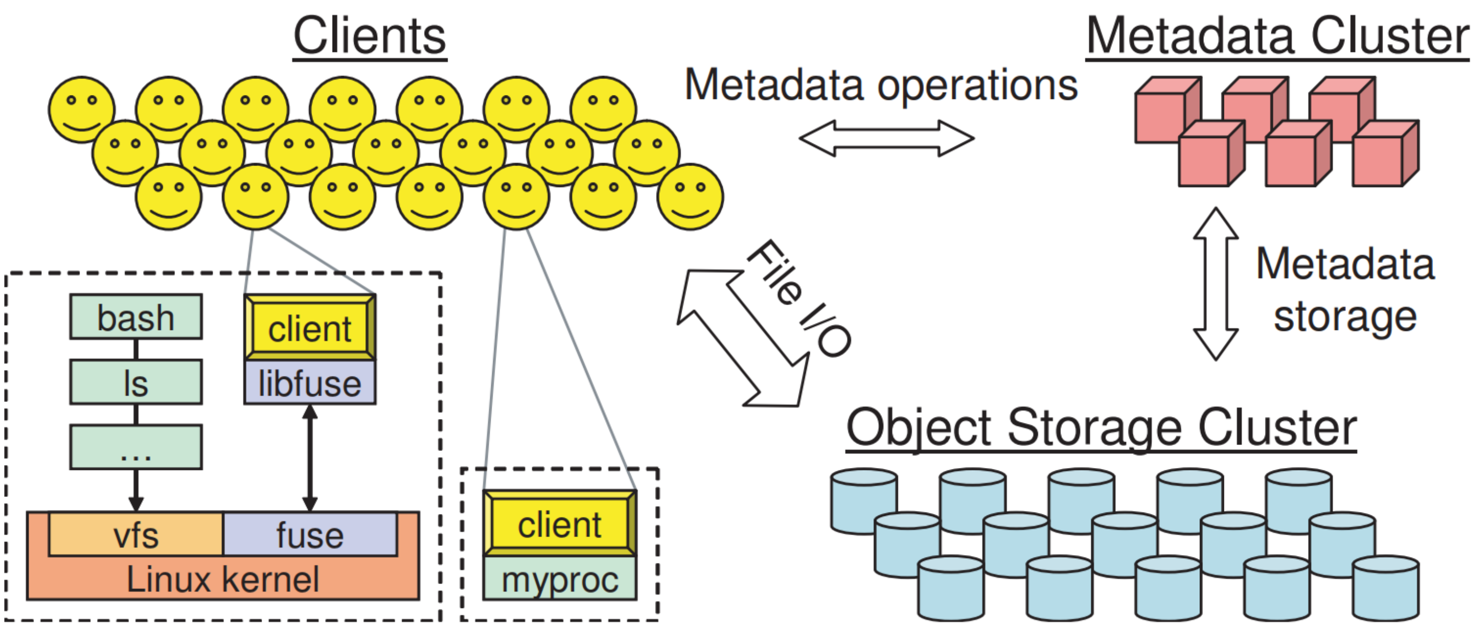
# Motivation

- **Problems on existing DFSs**
  - None of previous solutions has the combination of scalability, adaptiveness and reliability
  - Usually, metadata workload serves as the major obstacle
- **HDDs being replaced with smart OSDs (object storage devices)**
  - Great potential to improve scalability by distributing the complexity around data management to numerous nodes
  - But, reluctance to fully exploit intelligence of the OSDs...
    - Still relies on traditional file system principles.
    - Little or no distribution of workload itself.
- **Ceph is new distributed file system to resolve this issue**
  - Attempt to improve scalability along with adaptiveness and reliability by decoupling data and metadata

# System Overview

# System Overview

- **3 main components**
  - OSD cluster, MDS cluster and Clients.
- **Primary goals**
  - Scalability, Performance and Reliability

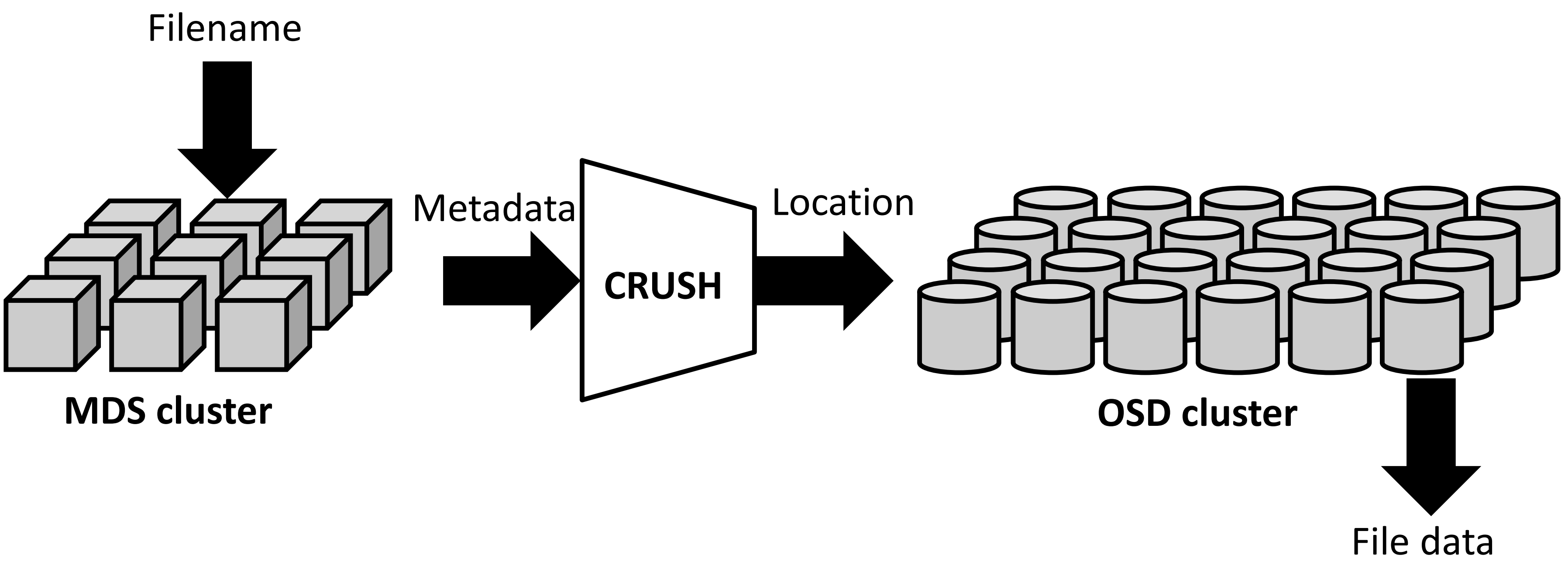


# System Overview

- **Decoupled data and metadata**
  - Calculate location rather than looking it up.
- **Dynamic distributed metadata management**
  - Dynamic Subtree Partitioning.
  - Efficiently utilize MDS cluster under any workload.
- **Reliable automatic distributed object storage**
  - OSD cluster is responsible for data migration, replication and failure detection/recovery.
  - MDS doesn’t care about OSD’s state.

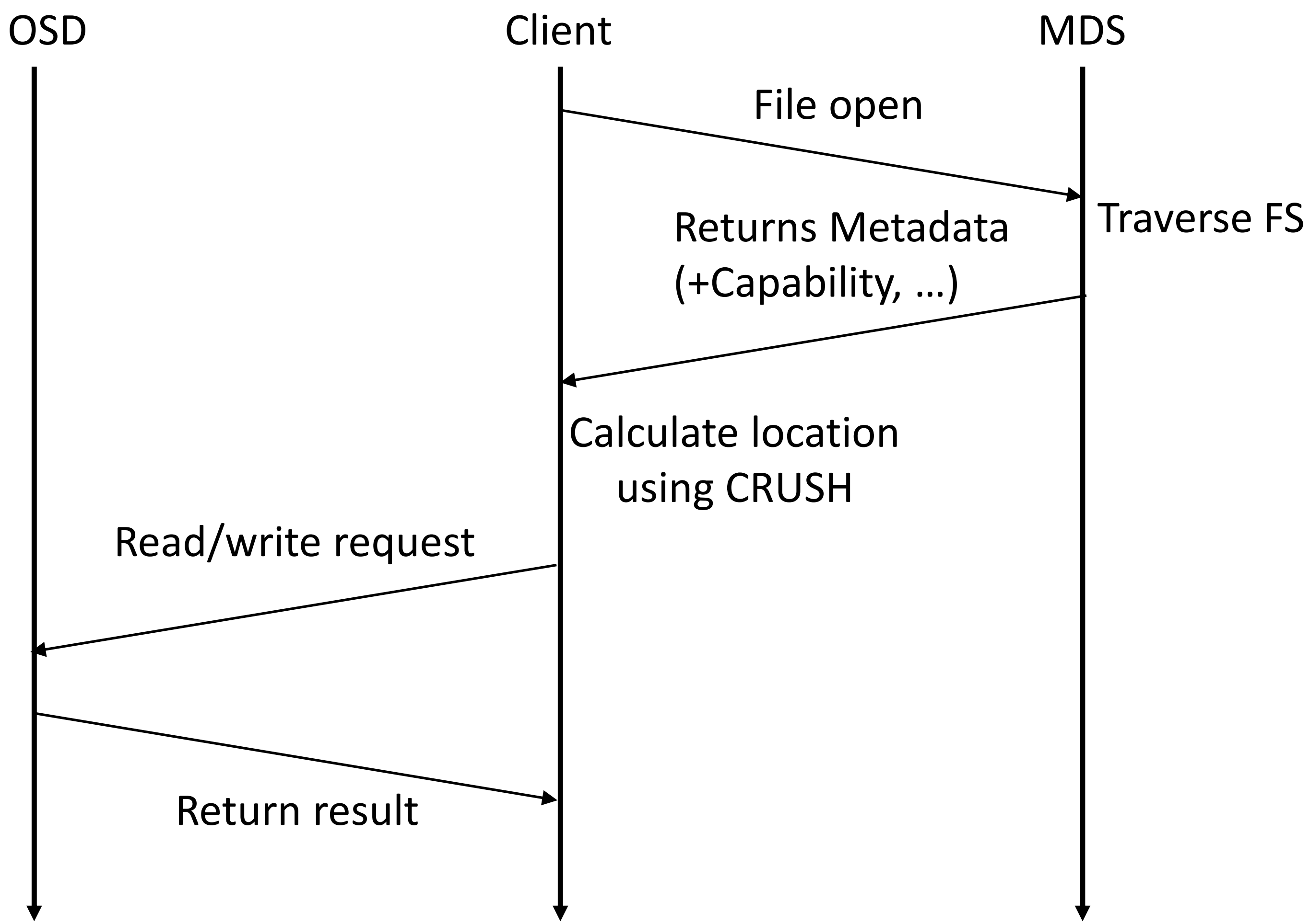
# Decoupled Data and Metadata

- **Calculate file's location rather than looking it up.**
  - Metadata storage has minimal metadata of file(80 bytes)
  - File's location can be calculated from minimal metadata using CRUSH (Controlled Replication Under Scalable Hashing).



# Decoupled Data and Metadata

- **Example read/write operation.**

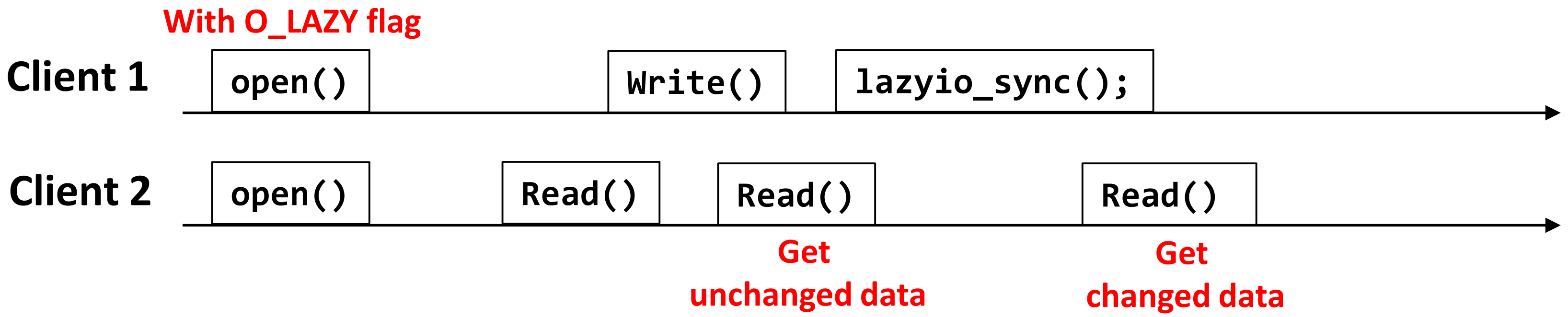


# Synchronization

- **POSIX semantics**
  - Reads have to reflect any previously written data.
  - Write is atomic.
  - However, this can be a performance killer for HPC workloads.
    - Read-write sharing a single large file

# Synchronization

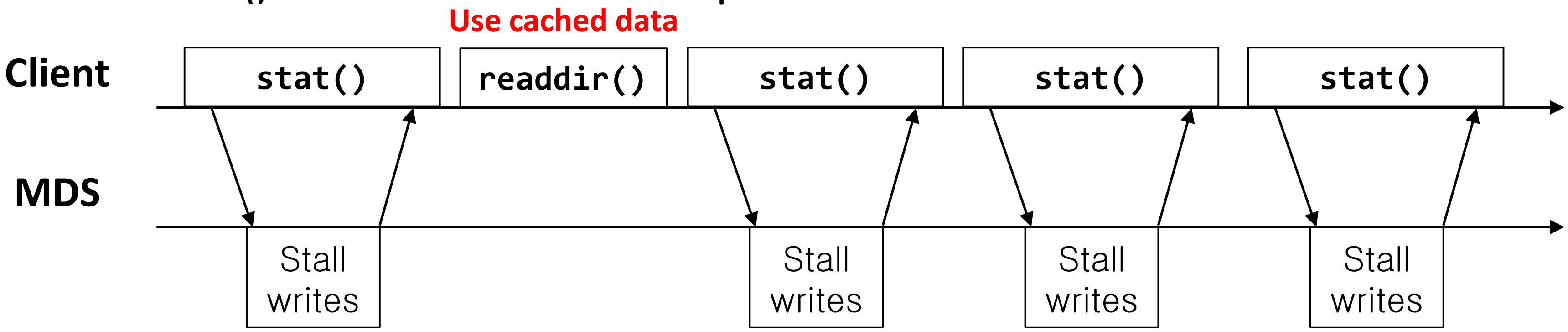
- **Relaxed coherence semantic**
  - Available with additional flag (O_LAZY) when opening a file.
  - Application will manage their own consistency.
  - Applications can explicitly synchronize with OSD using additional calls.
    - `lazyio_propagate()` flushes a given byte range to OSD.
    - `lazyio_synchronize()` will ensure that the effects of previous propagations are reflected in any subsequent reads.




* Not drawn to scale

# File system namespace operations

- **Caching directory data from stat() for following operations.**
  - Making common case fast.
  - Example: `readdir()` followed by `stat()` (`$>ls -l`).
  - Explicitly implemented as `readdirplus()` extension.
- **However, caching stat() data longer may behave incorrectly.**
  - Polling stat() may return inconsistent result in that case.
  - stat() will stall all writes to specified file and returns current state.

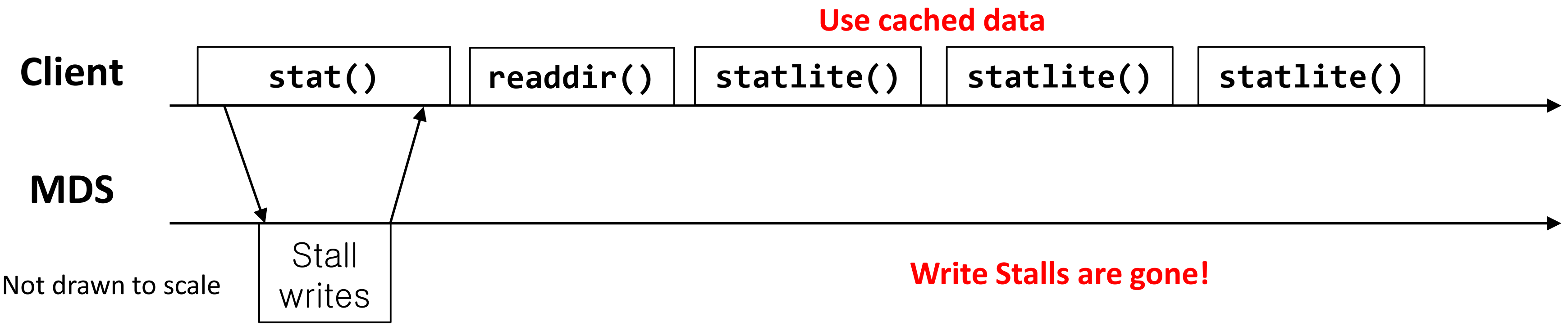



* Not drawn to scale

# File system namespace operations

- **Caching directory data from stat() for following operations.**
  - Making common case fast.
  - Example: `readdir()` followed by `stat()` (`$>ls -l`).
- **However, caching stat() data longer may behave incorrectly.**

- **statlite() can be employed if coherency is unnecessary.**




* Not drawn to scale

# Dynamically Distributed Metadata

# Dynamically Distributed Metadata

- **Light metadata (80 bytes)**
  - Directory entries and inodes., per
  - Allocation metadata is not necessary.
- **Simplified metadata workload**
  - Objects are distributed to OSDs using CRUSH and inode number.
  - Object location is **calculated** rather than looked up.
    - This will be covered in OSD section.

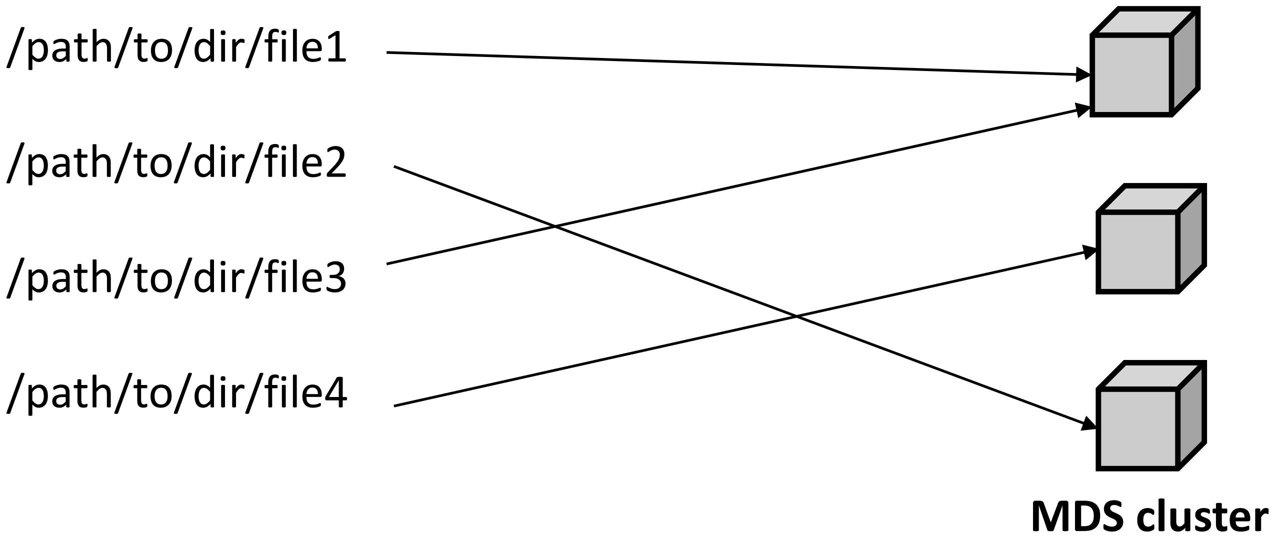
# Metadata Storage

- **Stored in OSD (diskless MDS) or Local disk.**
- **Per-MDS journaling**
  - Large, bounded and lazily flushed journal
  - Efficiently reduces disk writes by lazily flushing.
  - Ensures sequential write to maximize disk bandwidth.

- **However, Recovery scheme is not implemented.**
    - (At least at 2006)

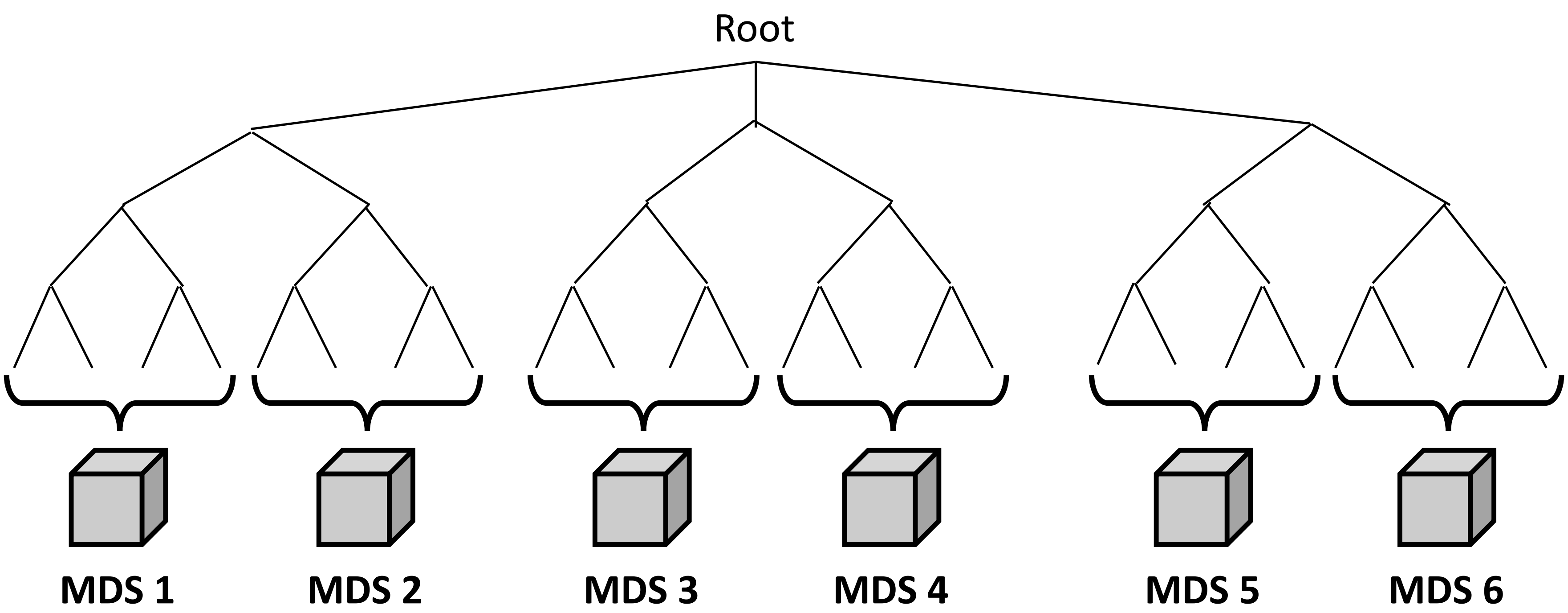
# Dynamic Subtree Partitioning

- **Issues on previous works**
  - Hash function effectively distributes workload with the cost of locality.



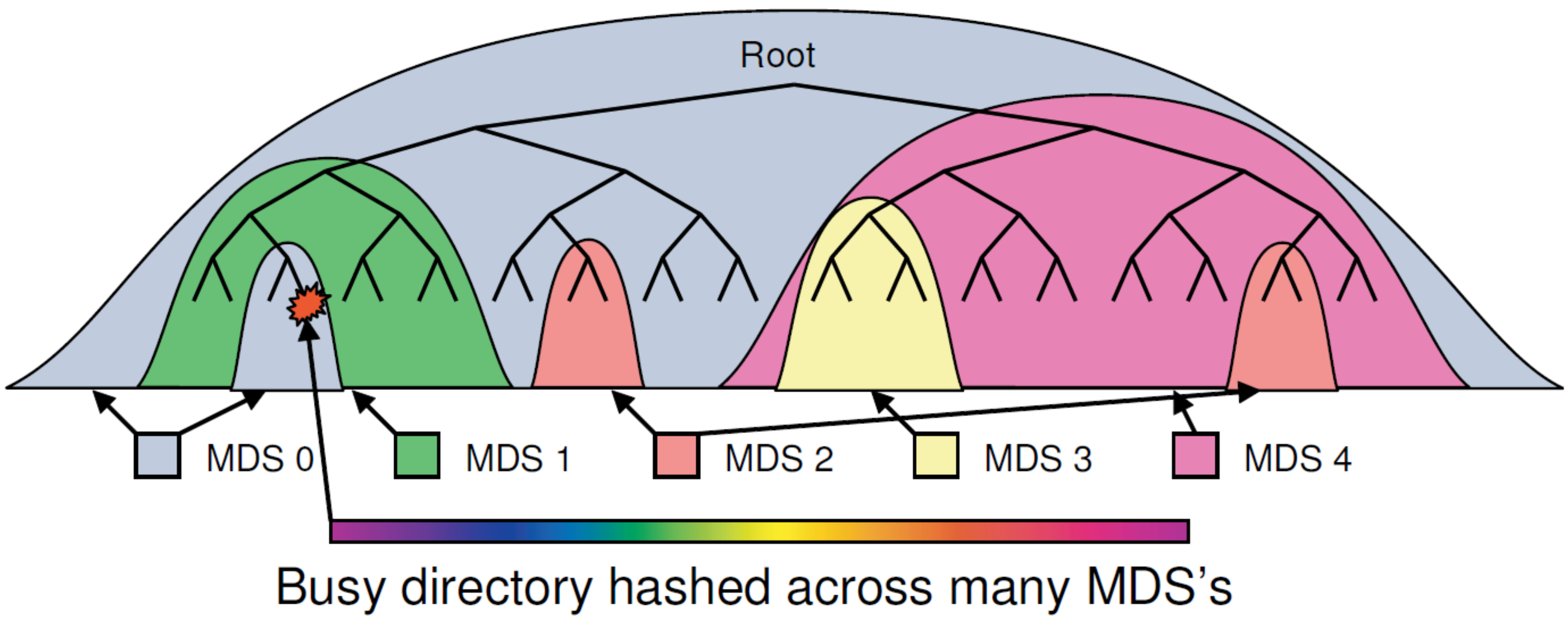
# Dynamic Subtree Partitioning

- **Issues on previous works**
  - static subtree partitioning shows high locality. But it is hard to cope with heavily skewed workload.



# Dynamic Subtree Partitioning

- **Compare and Balance**
  - Each MDS tracks load of itself and others'.
  - Compared periodically and evenly distributed across the cluster.

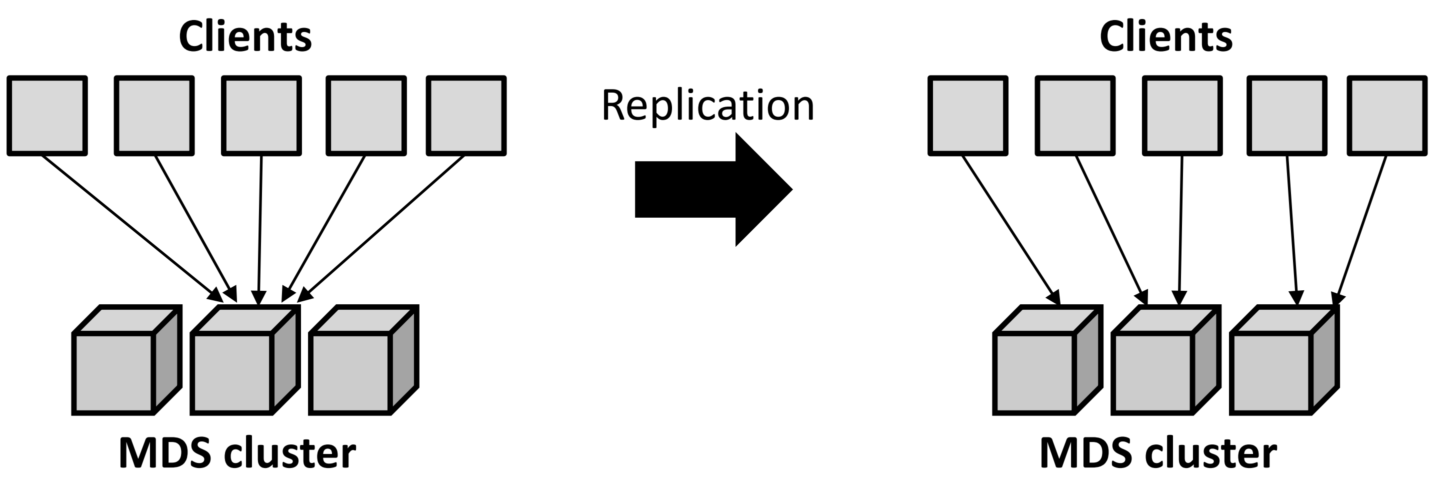


# Traffic control for hot spots

- **Heavily read directories' metadata**

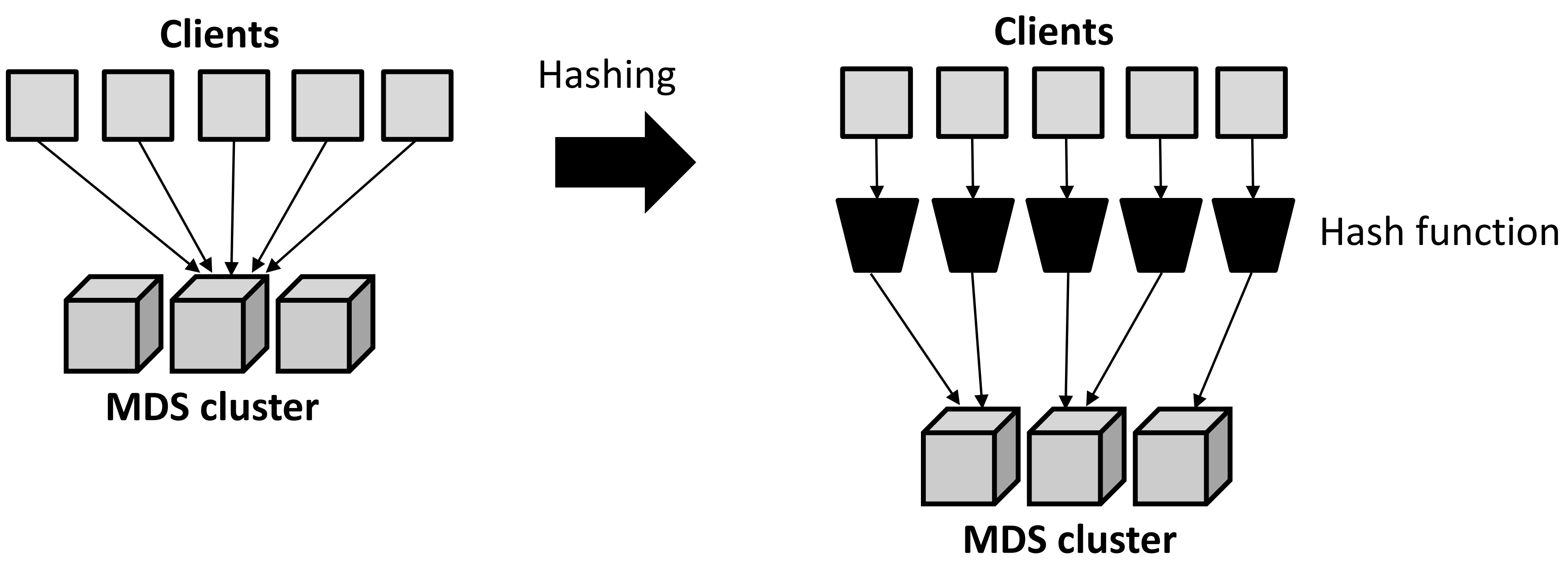
  (Example: Opening many files)

  - Contents are replicated across the cluster.
  - Load is distributed to other MDSs.



# Traffic control for hot spots

- **Heavily written directories' metadata**
  (Example: Creating many files)
  - Contents are hashed by file name and distributed across the cluster.
  - Sacrifices locality, but better scalability.



# Distributed Object Storage (Reliable Autonomic Distributed Object Store)

# RADOS

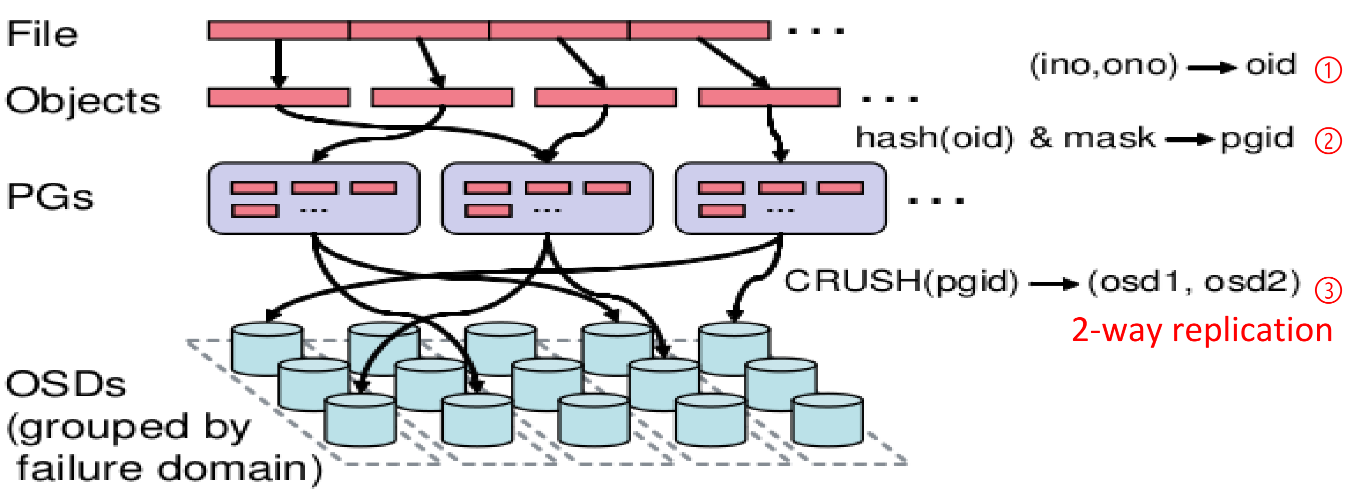
- **Distributed management**
  - Object replication, low-level allocation, cluster expansion, failure detection, recovery and other management operations are done by intelligent OSDs in a distributed manner
  - Rejecting any central server results in high scalability
  - Largely fueled by a data distribution function (CRUSH) which replaces allocation map.

# Data Distribution with CRUSH

- **Load balancing is important!**
  - Load asymmetry leads to ineffective utilization of storage bandwidth
- **Stochastic approach**
  - Distribute new data randomly
  - Migrate a random sample of existing data to new devices
  - Uniformly re-distribute data from removed devices

# Data Distribution with CRUSH (cont'd)

- **Distribution flow**
  - ① File striped into multiple objects
    - Object ID = {file inode, stripe number}
    - Simple combination
  - ② Objects grouped into placement groups (PGs)
    - PG ID = hash(Object id) & mask
    - Simple hash function and an adjustable bit mask
  - ③ PGs assigned to an ordered list of OSDs
    - OSDs = CRUSH(PG ID)
    - Pseudo-random mapping function '***CRUSH (Controlled Replication Under Scalable Hashing)***'



# Data Distribution with CRUSH (cont'd)

- **CRUSH**
  - Approximate a uniform probability distribution
  - Pseudo-random
  - **<u>Deterministic</u>**

    : Any system can calculate CRUSH function independently without consulting a central allocator

    NO metadata server required
  - Distribution controlled by *<u>cluster map & placement rules</u>*
    - **Cluster map**

      **Hierarchical, weighted map of storage devices.**

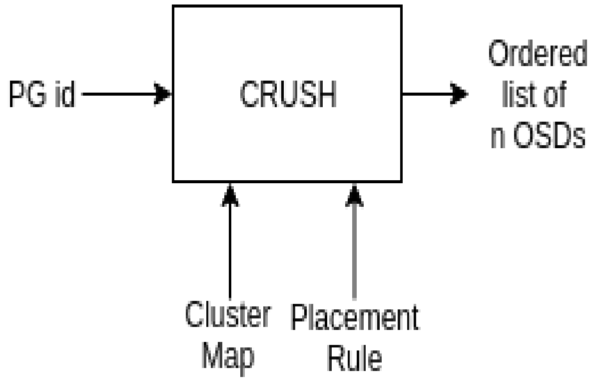
      **# of storage devices, capability of each device, organization of devices ...**
    - **Placement rules**

      **Level of replication (2-way, 4-way …)**

      **Constraints on placement (separate replicas across different failure domains)**

      **-> e.g) All replicas should be on different shelves**

      **in which devices share same power supply.**



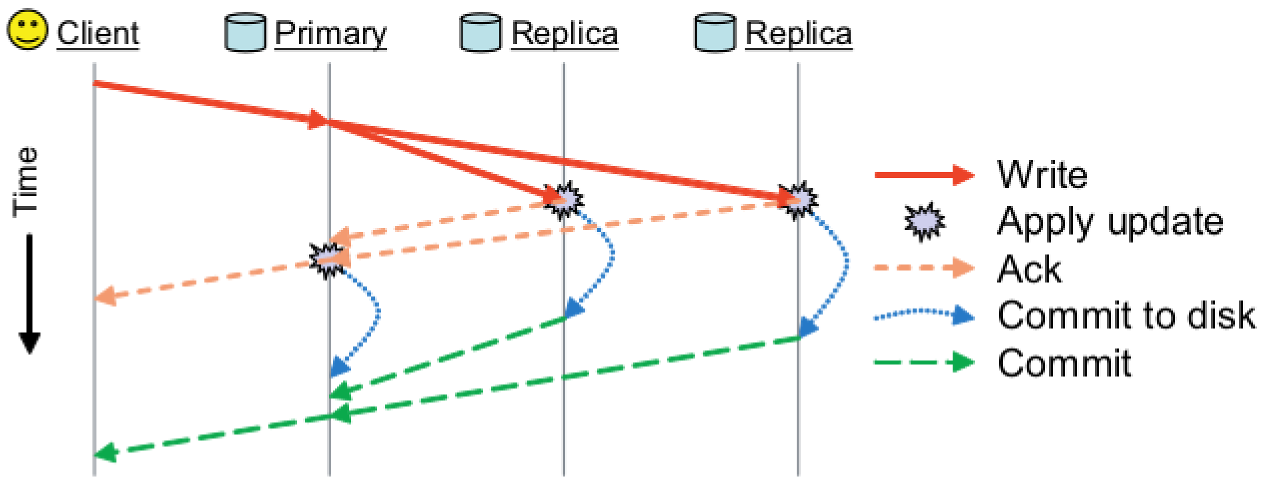
# Replication

- **Primary-copy based replication**
  - First non-failed OSD in a list of OSDs is primary copy
  - Primary copy forwards write to the replicas
  - Client does not need to care about replicas
  - No bandwidth burden on the client due to replication

# Data Safety

- **Data safety achieved by update process**
  - Send *Ack* to client once all replicas have received the update
  - Send *Commit* once all replicas have committed update to disk
  - Clients buffer write until they get commit,

    and replay in the case of failure

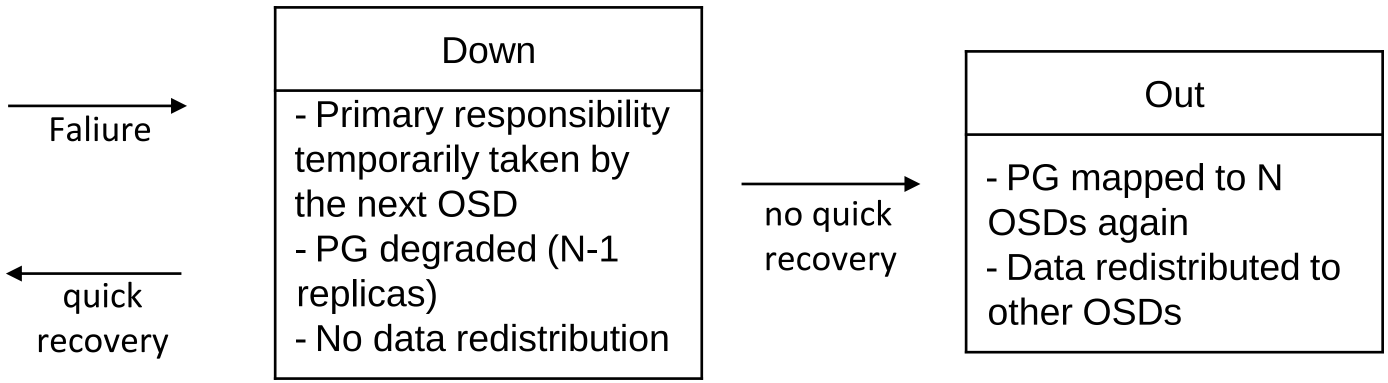


# Failure Detection

- **Active failure detection**
  - Failures that make an OSD unreachable require active monitoring
  - Each OSD monitors those peer OSDs with which it shares PGs

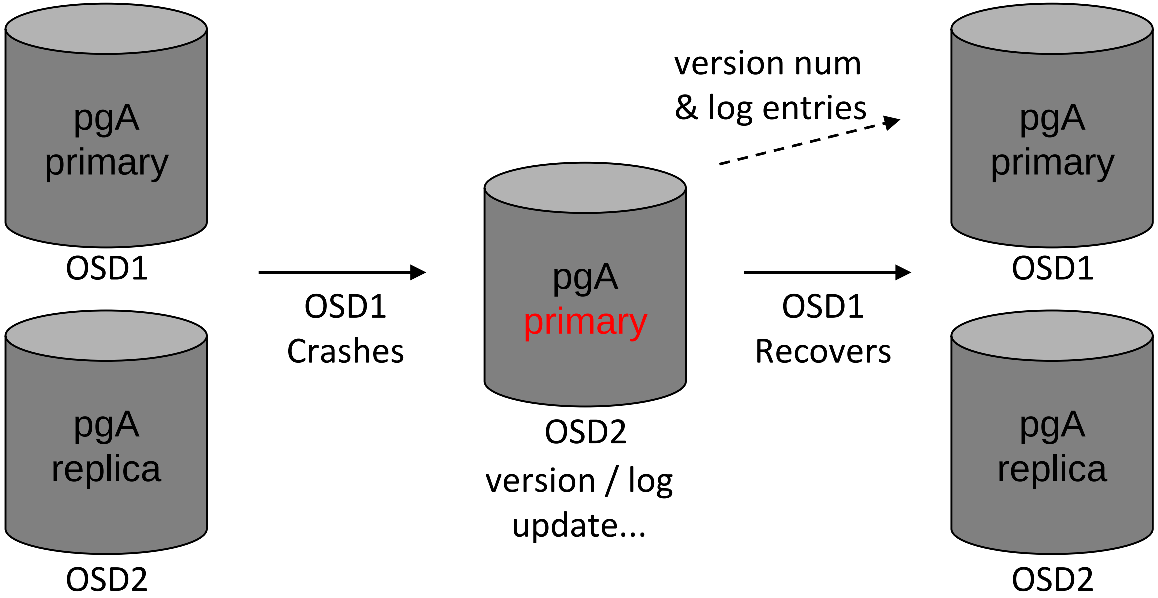
    -> Distributed monitoring allows fast detection
  - A small cluster of monitors centrally collects anomalies and maintain synchronized cluster map
  - A unresponsive OSD is initially marked ***down*** for a specific length of time, and marked ***out*** later if quick recovery is not available

    -> Distinction between **'Down'** and **'Out'** avoids hasty data replication



# Recovery and Cluster Update

- **Failure recovery driven by individual OSDs**
  - OSDs maintain a version number for each object and a log for each PGs
  - When OSD receives an updated cluster map, and a PG's membership has changed,
    - for primary PGs, OSD collects current replicas' PG versions to determine correct PG contents
    - for replicated PGs, OSD sends the primary its current PG version



# EBOFS

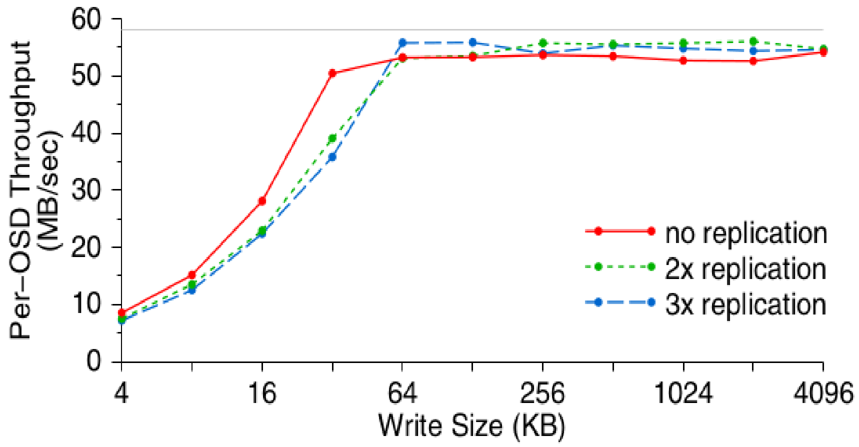
- **EBOFS (Extent and B-tree based Object File System)**
  - Existing general purpose local file system is not suitable
    - Existing kernel interface limits RADOS's ability to understand safe commit timing
    - Journaling accompanies big performance penalty
    - POSIX interface fails to support atomic data & metadata update
  - Each Ceph OSD manages its local object storage with EBOFS
    - Fully integrated B-tree service
    - Block allocation done in terms of extent (start, length)
    - Free space sorted by size and location
    - Aggressive copy-on-write

# Performance and Scalability Evaluation

# Data Performance (Throughput)

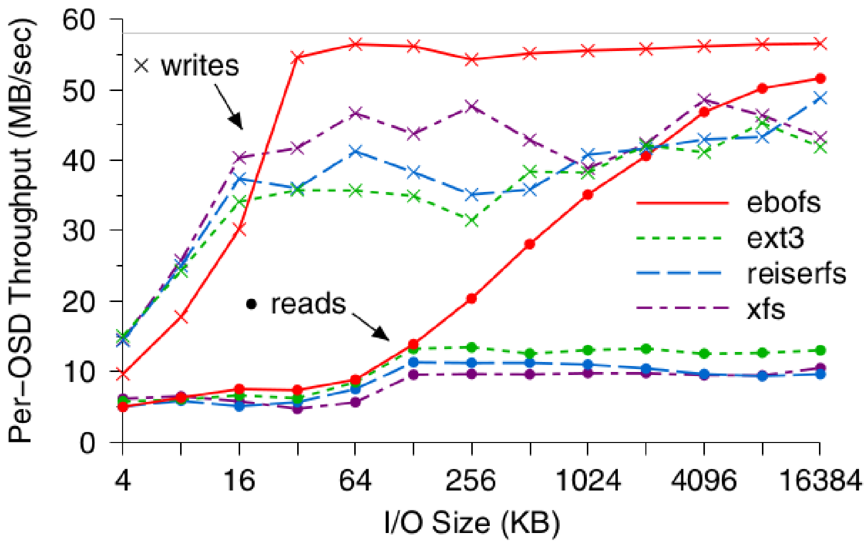
- **Per-OSD throughput with varying write sizes and replication.**

  ***Replication has minimal impact on OSD throughput.***




- **Performance of EBOFS compared to general-purpose file systems.**

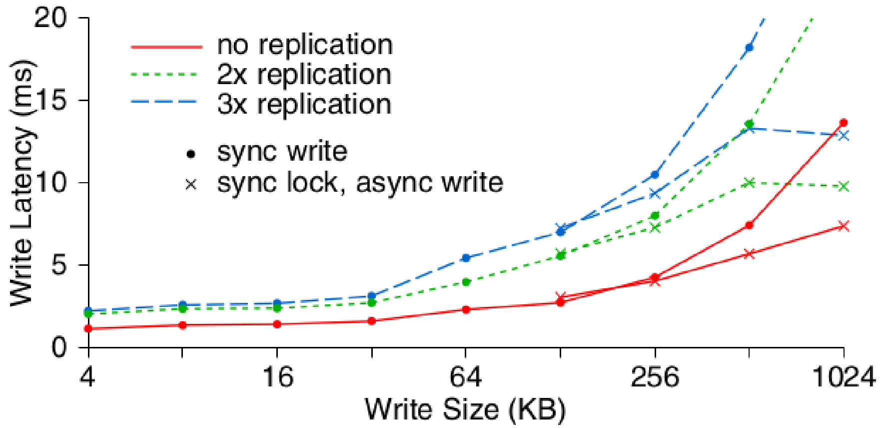
  ***Small writes suffer from coarse locking, but it nearly saturates the disk bandwidth for writes larger than 32KB.***



# Data Performance (Latency & Scalability)

- **Write latency for varying write sizes and replication**

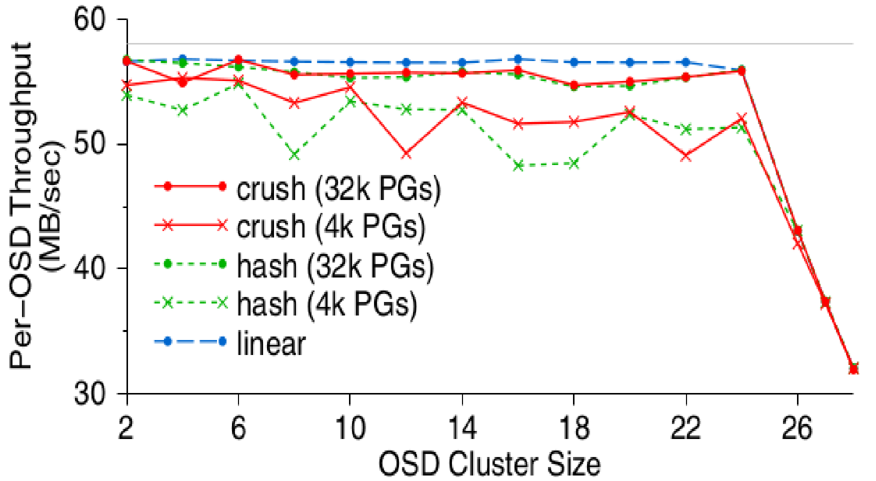
  ***Retransmission overhead dominates for large writes.***




- **Per-OSD write throughput with the increasing size of the cluster and different distribution schemes**

  ***Linear striping is good, but subject to failure or cluster changes.***

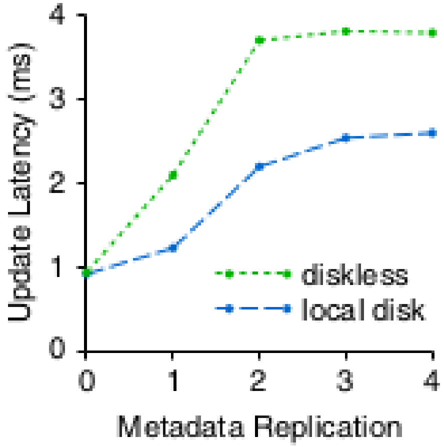
  ***Better throughput for CRUSH/hash with more PGs. (more uniform distribution)***



# Metadata Performance (Latency)

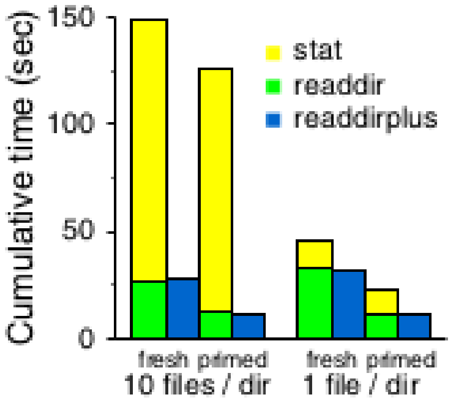
- Metadata <u>update</u> latency for an MDS with and without a local disk for varying replication

  *Using local disk lowers update latency.*




- Metadata <u>read</u> latency during a file system walk (*readdir* followed by *stat*)

  *readdir time reduces due to MDS cache, and readdirplus (relaxed consistency) eliminates time for stat.*

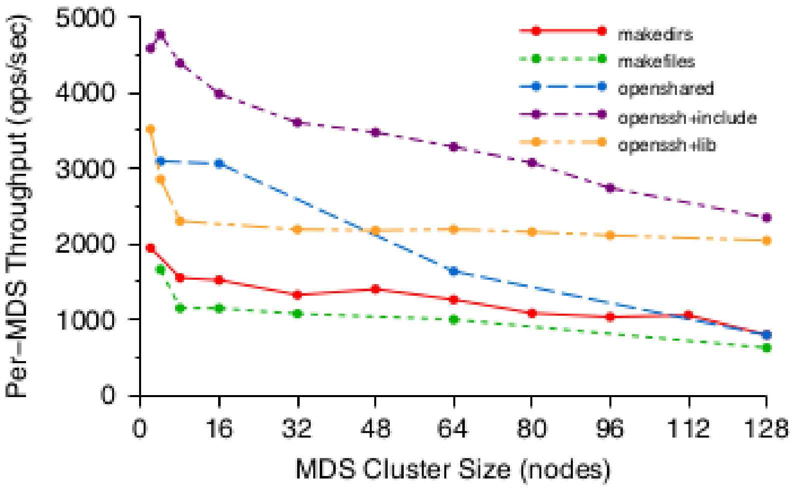


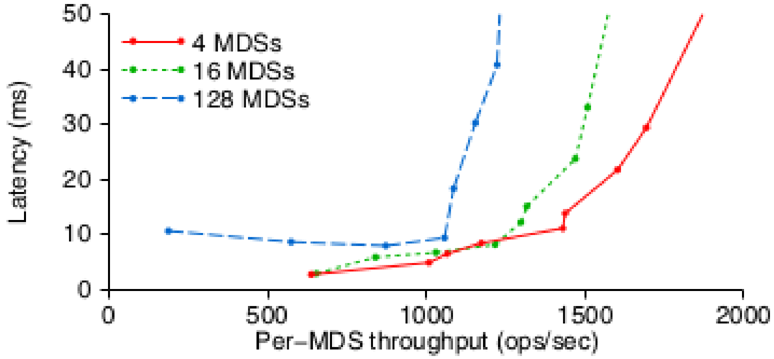
# Metadata Performance (Scalability)

- **Per-MDS throughput with the increasing cluster size**

  ***Not perfect linear scaling, but no more than 50% below.***

  ***Load imbalance increases with the cluster size, which imposes limits on scalability.***



# Summary

# Summary

- **Data - Metadata Separation**
  - CRUSH: enables independent object location calculation of client
  - Usually, metadata workload serves as the major obstacle
- **MDS optimization**
  - Load balancing by dynamic subtree partitioning / hot spot replication
- **OSD optimization (RADOS)**
  - Distributed / autonomous allocation, replication, failure detection and recovery
  - EBOFS, optimal local file system for Ceph

Scalability , High-Performance, Reliability!