Peta-scale Data Storage Research at UCSC

Scott A. Brandt
Darrell D. E. Long
Sage Weil
Feng Wang
Andrew Leung
Ethan L. Miller
Carlos Maltzahn
Qin Xin
RJ Honicky
Joel Wu

University of California, Santa Cruz

Sponsored by LANL, Sandia, LLNL

Peta-scale Data Storage Challenges

- Massive scale of everything
  - Huge files, directories, data transfers, etc.
- Managing the data
  - Coordinating the activity of thousands of disks
- Managing the metadata
  - Unified namespace
- Workload
  - Scientific and general purpose workloads
- Dynamic capacity
  - Must be able to grow (or shrink) dynamically
- Reliability
  - Thousands of hard drives $\Rightarrow$ frequent failures
- Security
  - Authentication, encryption, etc.
- Performance
  - Hot spot avoidance
  - Many possible bottlenecks
- Quality of Service
  - Guaranteed performance with mixed workloads
- Usability
  - Finding anything among all of that information

Ceph

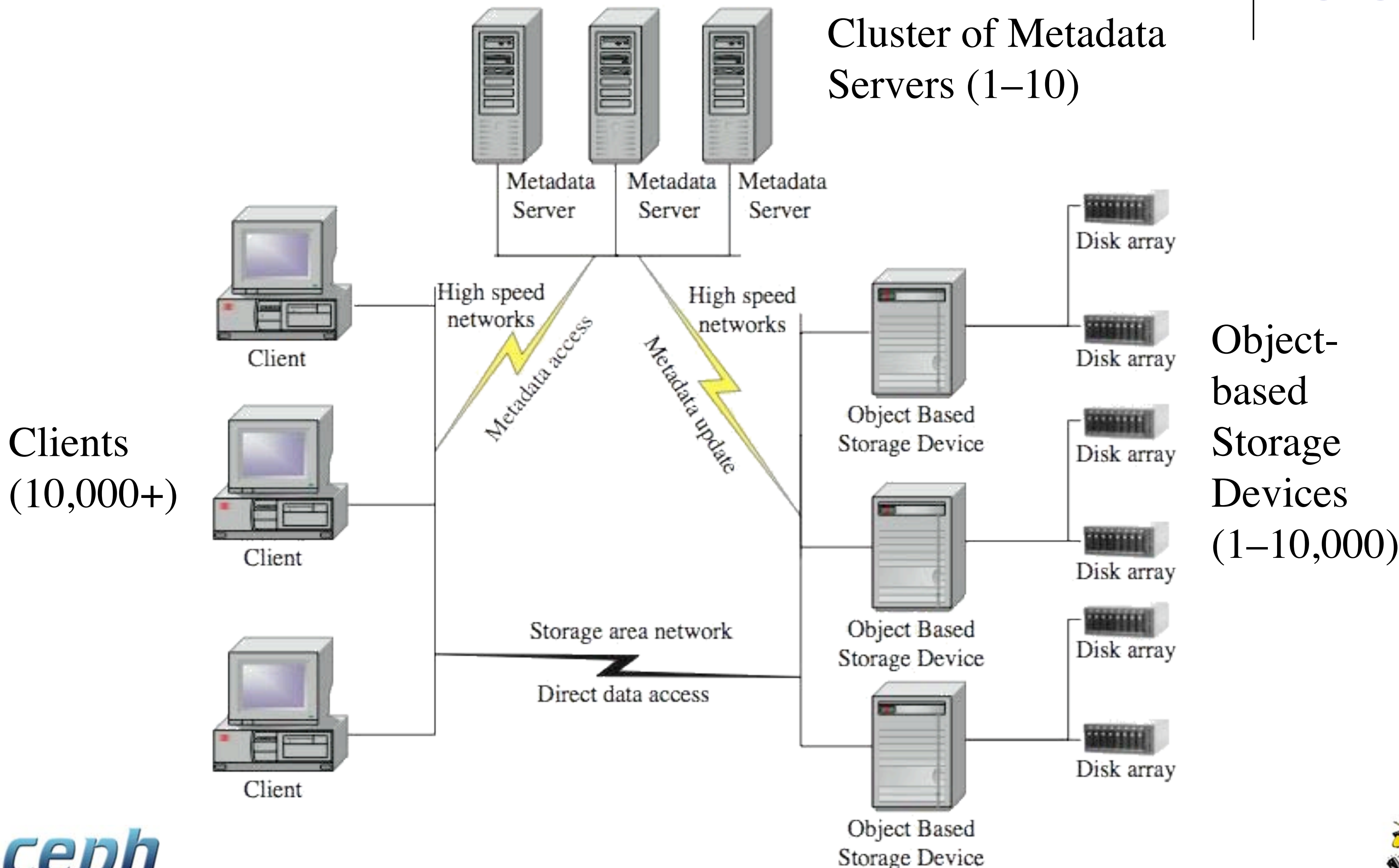
Usage

- POSIX-like interface
  - Standard file/directory semantics
- High-performance direct access from 100,000+ clients, to
  - Different directories, same directory, same file
- Mid-performance local access by visualization workstations w/QoS
- Wide-area general-purpose access

Performance

- 20 PB storage system
  - 1–10,000 hard drives
- 1 TB/sec aggregate throughput
  - 1,000–10,000 hard drives pumping out data as fast as they can
- Billions of files
  - Bytes to terabytes
  - 1–100,000+ files/directory
- Very low-latency metadata

Ceph Architecture



Ceph Research

<u>Metadata Cluster Management</u>

1. Lazy Hybrid
2. Dynamic Subtree Partitioning

<u>Client SW</u>

1. Interface
2. Cache Mgmt
3. Workload characterization

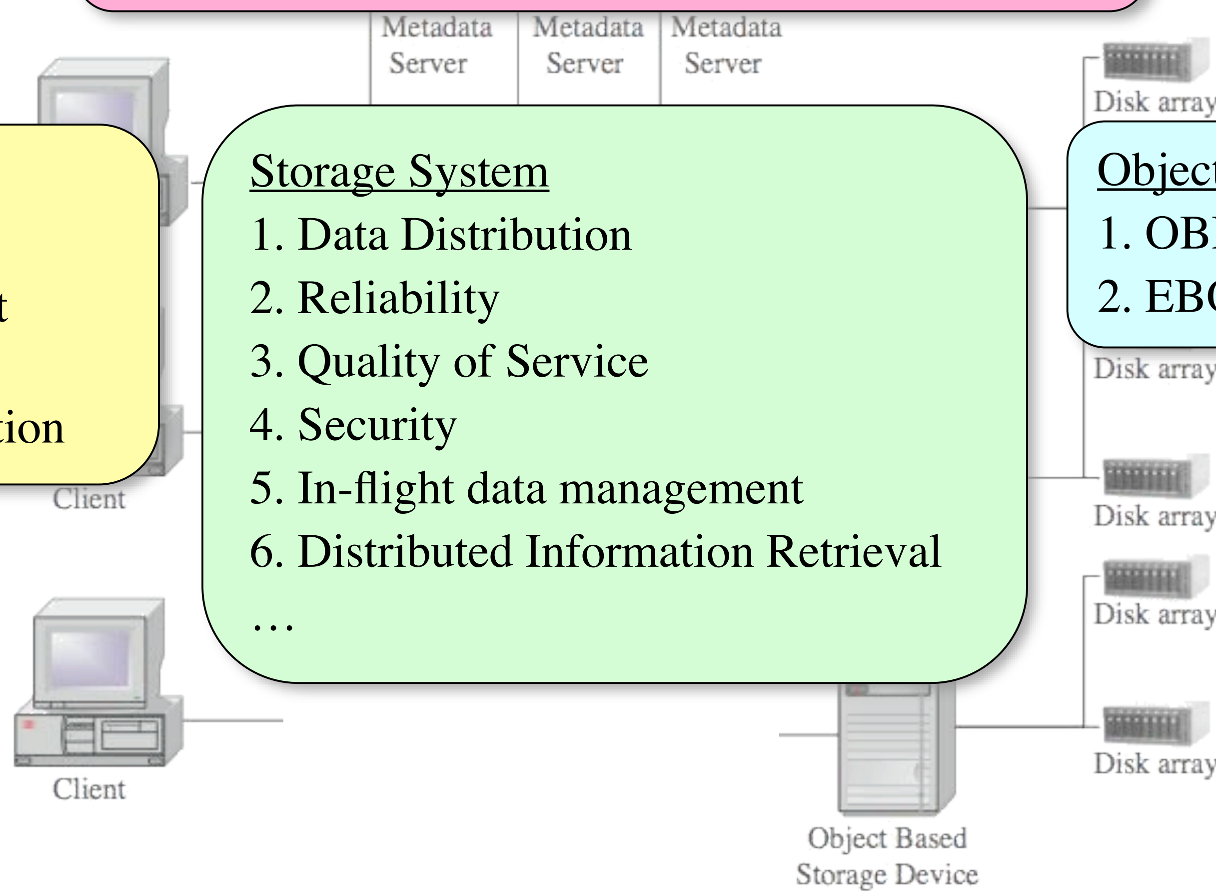
<u>Storage System</u>

1. Data Distribution
2. Reliability
3. Quality of Service
4. Security
5. In-flight data management
6. Distributed Information Retrieval

…

<u>Object Storage</u>

1. OBFS
2. EBOFS



Today: Ceph Overview (Sage)

- Client operation
  - System overview
- Ceph components
  - CRUSH pseudo-random data placement
  - DSP distributed metadata management
  - RADOS reliable, distributed object storage
  - EBOFS high-performance object storage
- Evaluation
  - Prototype performance numbers

Ceph: A Scalable, High-Performance Distributed File System

Sage Weil

University of California, Santa Cruz

Ceph—
Key Design Principles

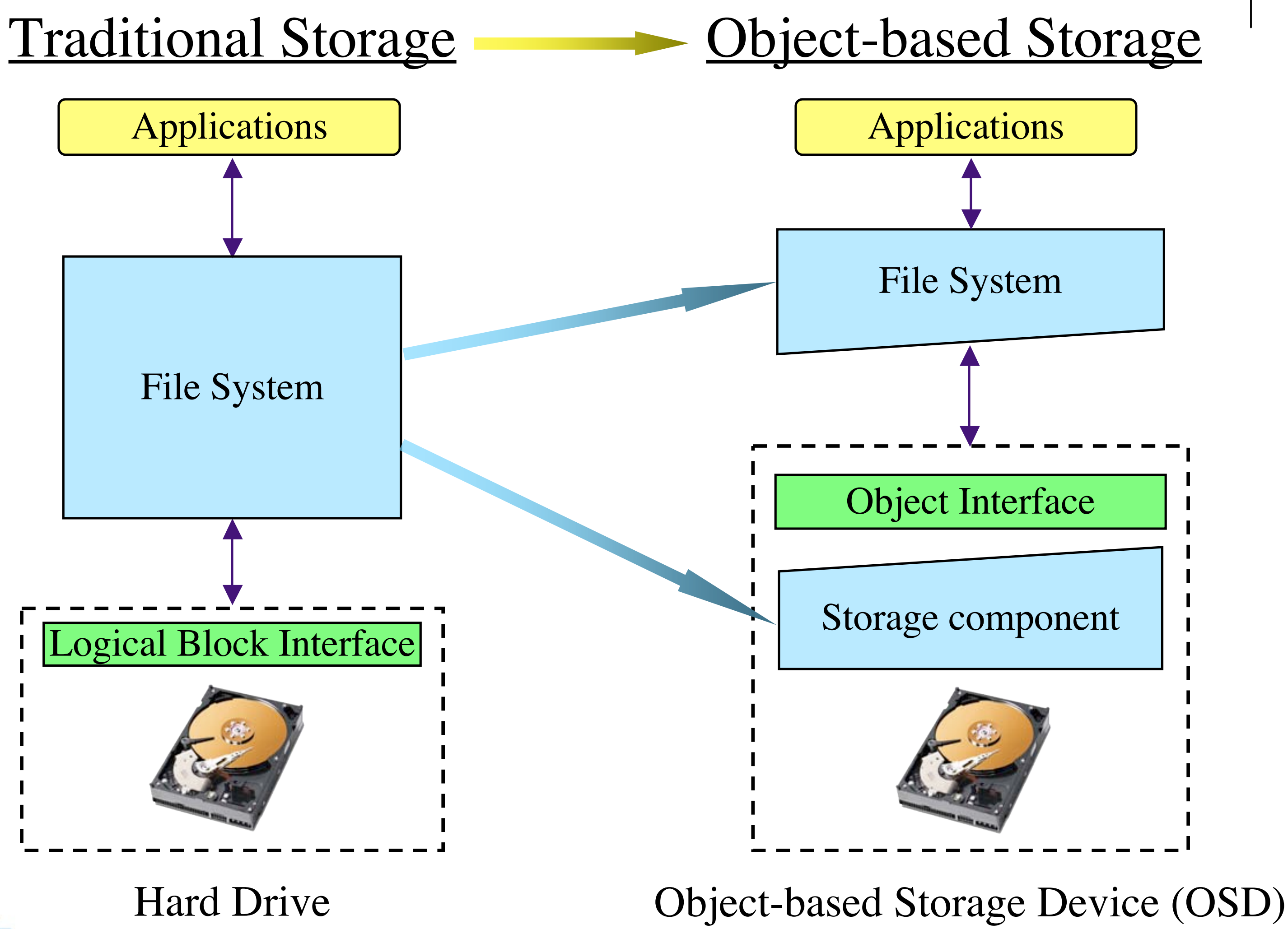
- Maximal separation of data and metadata
  - Object-based storage
  - Independent metadata management
  - CRUSH – data distribution function
- Dynamic metadata management
  - Adaptive and scalable
- Intelligent disks
  - Reliable Autonomic Distributed Object Store

ceph
PETABYTE SCALE STORAGE

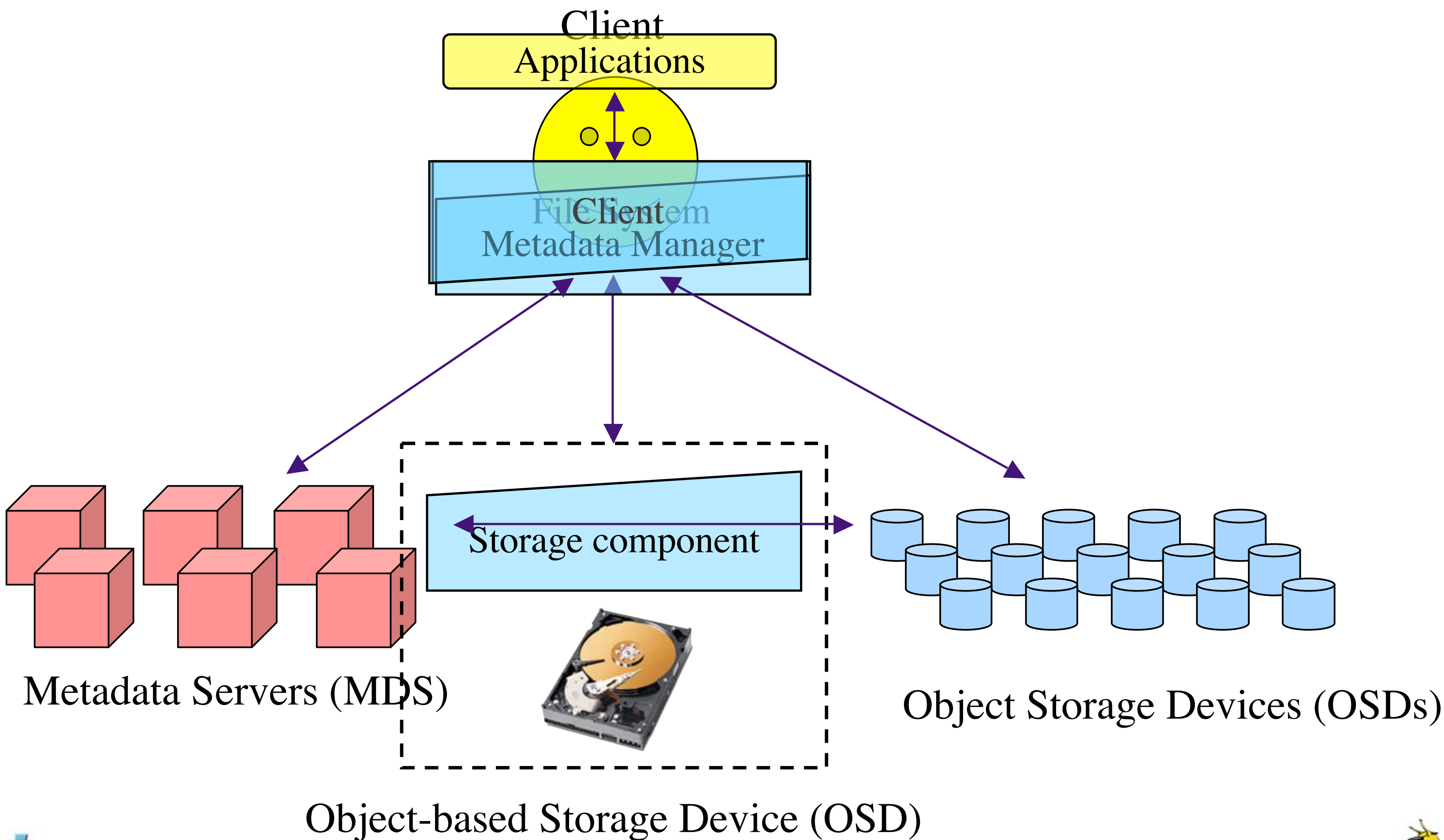
Outline

- Maximal separation of data and metadata
  - Object-based storage
  - Independent metadata management
  - CRUSH – data distribution function
- Dynamic metadata management
  - Adaptive and scalable
- Intelligent disks
  - Reliable Autonomic Distributed Object Store

Object-based Storage Paradigm



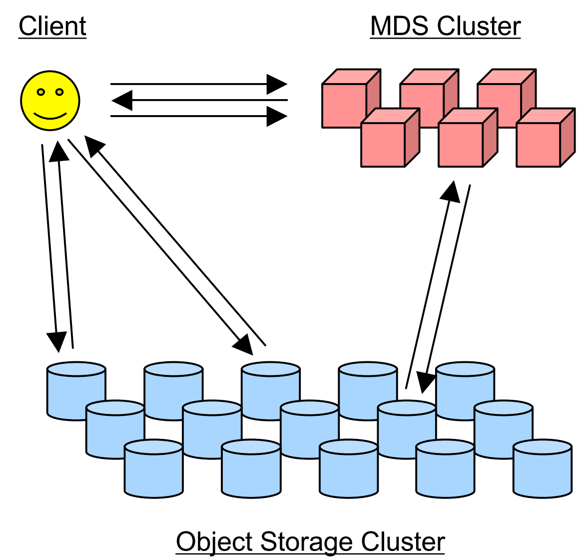
Ceph—
Decoupled Data and Metadata



ceph
PETABYTE SCALE STORAGE

Ceph— A Simple Example

- fd=open("/foo/bar",O_RDONLY);
  1. Client: requests open from MDS
  2. MDS: reads directory "/foo" from OSDs
  3. MDS: issues "capability" for "/foo/bar"
- read(fd,buf,10000);
  4. Client: **calculates** name(s) and location(s) of data object(s)
  5. Client: reads data from OSDs
- close(fd);
  6. Client: relinquishes capability to MDS



- MDS stays out of I/O path
- Client doesn't need to look up the location of file data

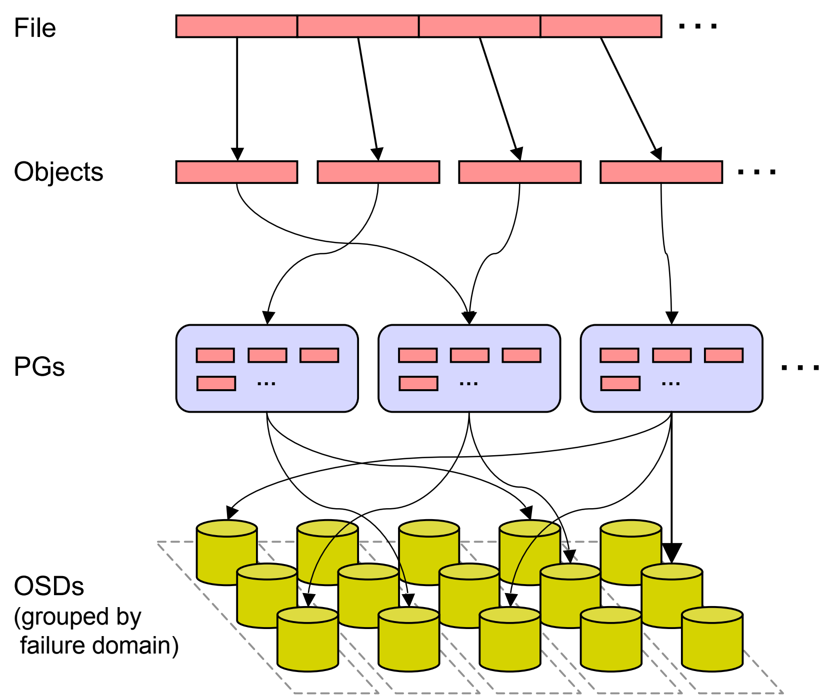
CRUSH—
Simplifying Metadata

- Conventionally
  - Directory contents (filenames)
  - File inodes
    - Ownership, permissions
    - File size
    - Block list
- CRUSH
  - Small “map” completely specifies data distribution
  - Eliminates allocation lists
  - Inodes “collapse” back into small, almost fixed-sized structures
    - Embed inodes into directories that contain them
    - No more large, cumbersome inode tables

CRUSH—
Data distribution with a *function*

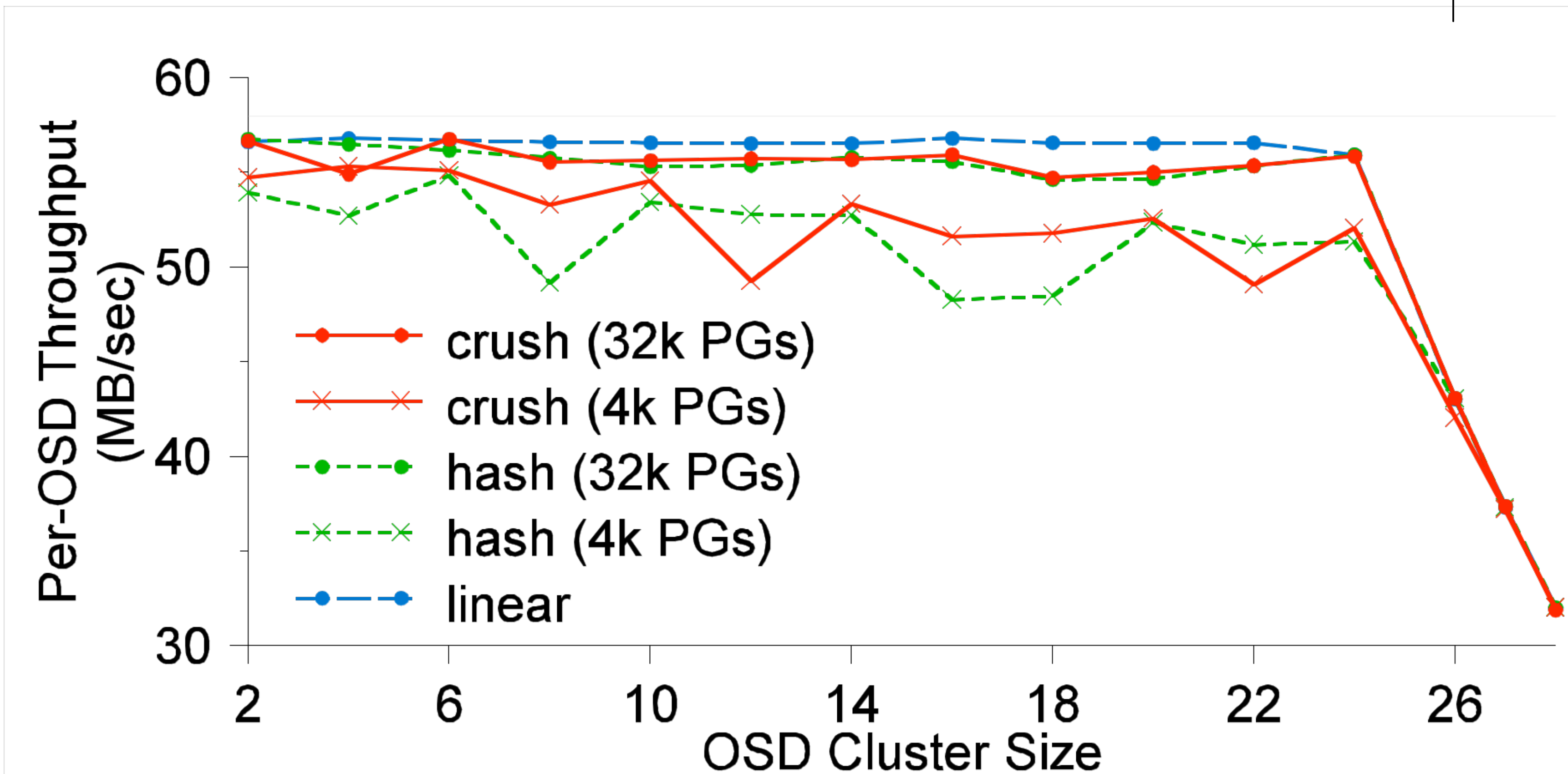
- Files striped across many *objects*
  - Striping strategy specified in inode
  - object_id = <inode_num, object_num>
- Objects mapped to *placement groups* (PGs)
  - pg_id = hash(object_id) & mask
- CRUSH maps PGs to OSDs
  - Pseudo-random distribution
  - Statistically uniform
  - Replicated on multiple OSDs
- CRUSH is…
  - A function—calculable everywhere (*no explicit tables*)
  - Stable—adding/removing OSDs moves few objects
  - Reliable—replicas span failure domains

  - ...everything you'd normally want to do using conventional allocation tables!



ceph PETABYTE SCALE STORAGE

OSD Cluster Scaling—CRUSH vs careful striping

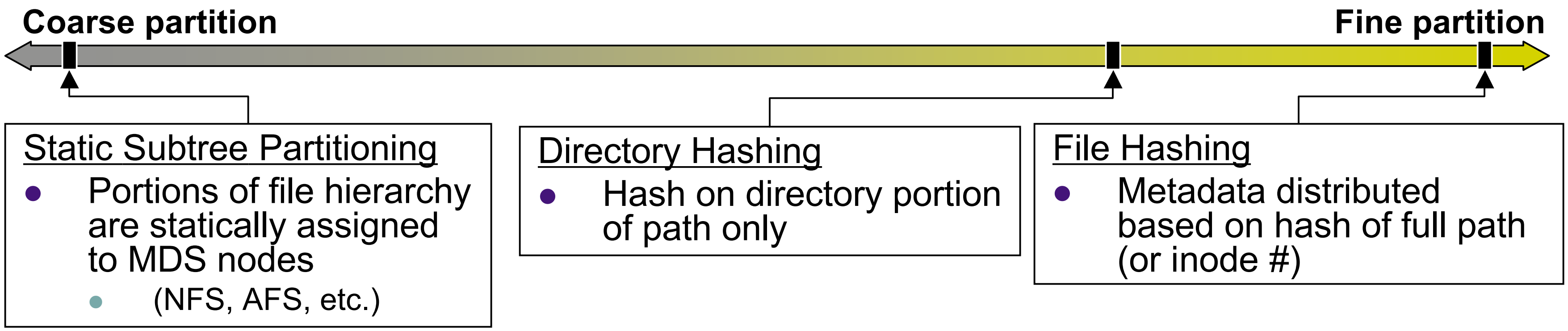


- Higher placement group count reduces statistical variance, divergence from optimal (write throughput shown)

Outline

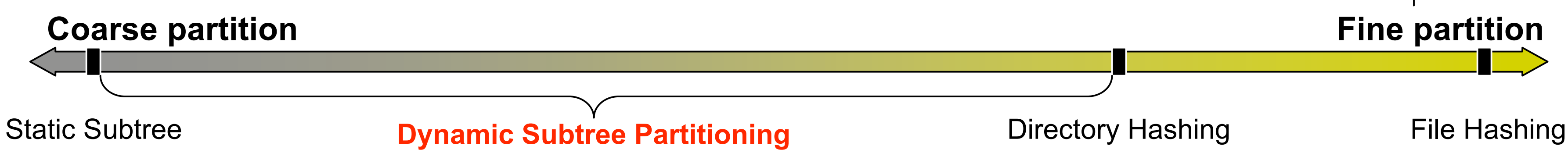
- Maximal separation of data and metadata
  - Object-based storage
  - Independent metadata management
  - CRUSH – data distribution function
- Dynamic metadata management
  - Adaptive and scalable
- Intelligent disks
  - Reliable Autonomic Distributed Object Store

Metadata—Traditional Partitioning



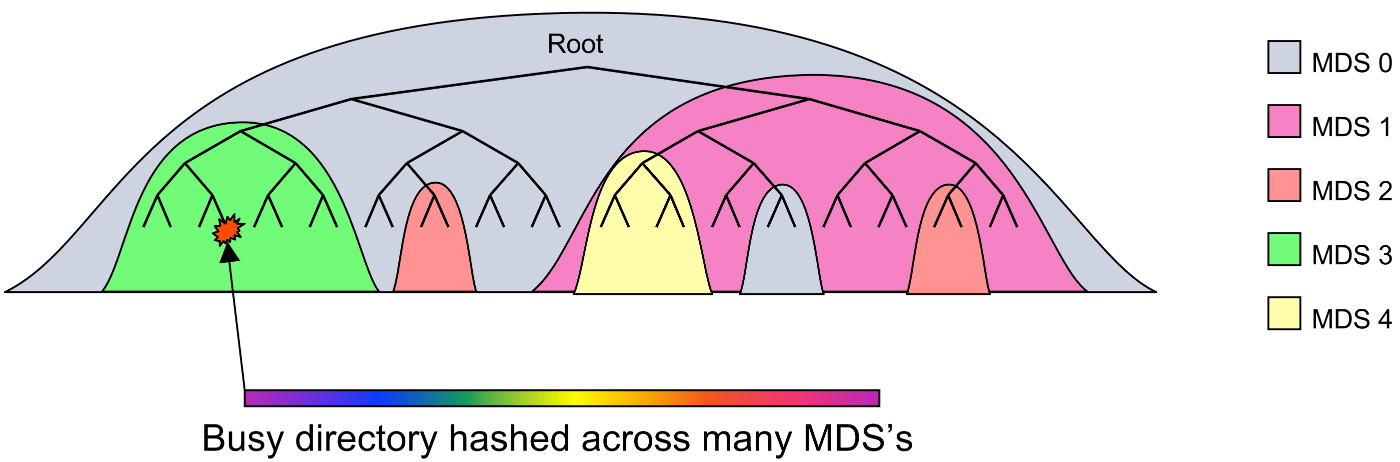
- Coarse distribution (static subtree partitioning)
  - hierarchical partition preserves locality
  - high management overhead: distribution becomes imbalanced as file system, workload change
- Finer distribution (hash-based partitioning)
  - probabilistically less vulnerable to “hot spots,” workload change
  - destroys locality (ignores underlying hierarchical structure)

Ceph's Dynamic Partitioning



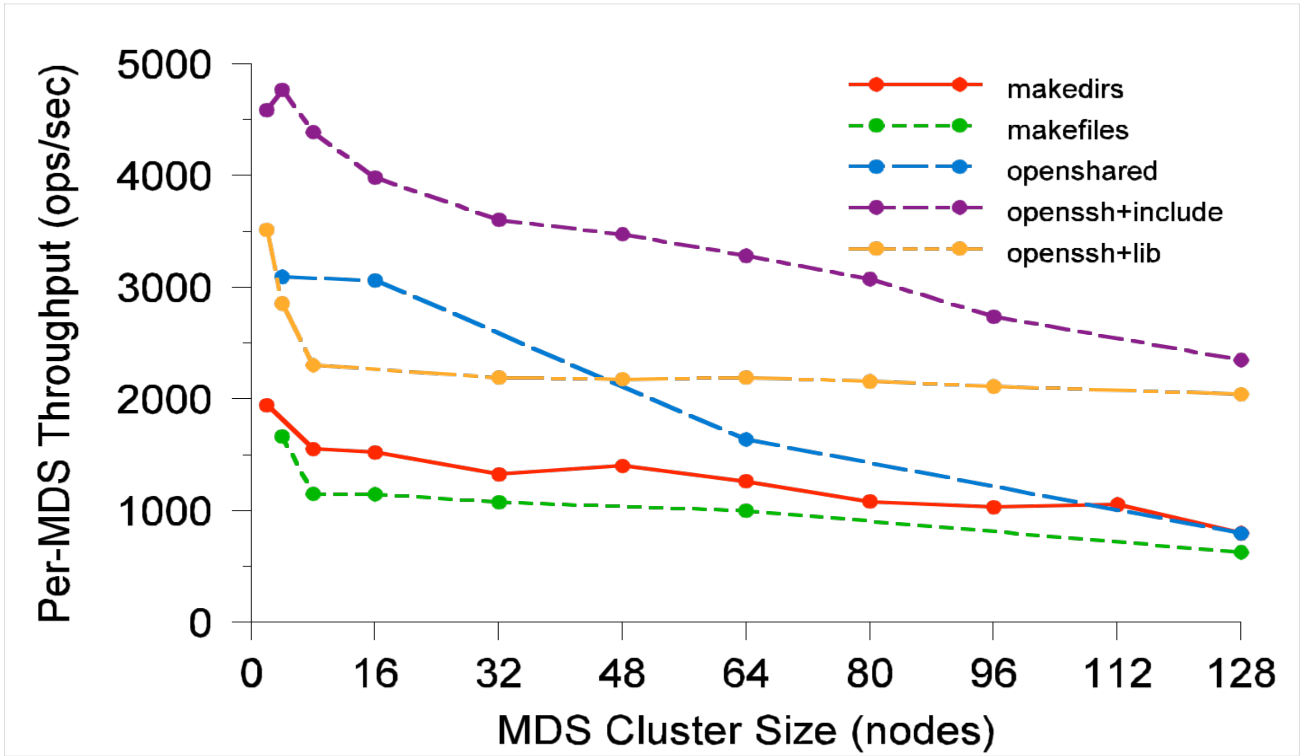
- Ceph dynamically distributes arbitrary subtrees of the hierarchy
  - Coarse partition preserves locality
  - Adapt distribution to keep workload balanced
    - Migrate subtrees between MDSs as workload changes
- Adapt distribution to cope with hot spots
  - Heavily read directories replicated on multiple MDSs
  - Heavily written directories individually hashed across multiple nodes

Metadata Partition



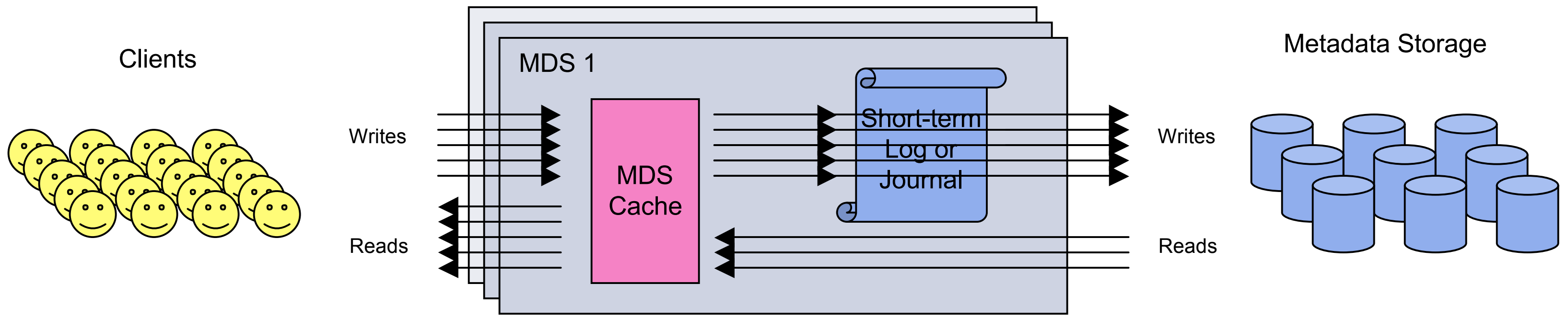
- Scalability
    - Arbitrarily partitioned metadata
- Adaptability
    - Cope with workload changes over time, and hot spots

Metadata Scalability



- Up to 128 MDS nodes, and 250,000 metadata ops/second
  - I/O rates of potentially many terabytes/second
  - Filesystems containing many petabytes (or exabytes?) of data

Metadata Storage

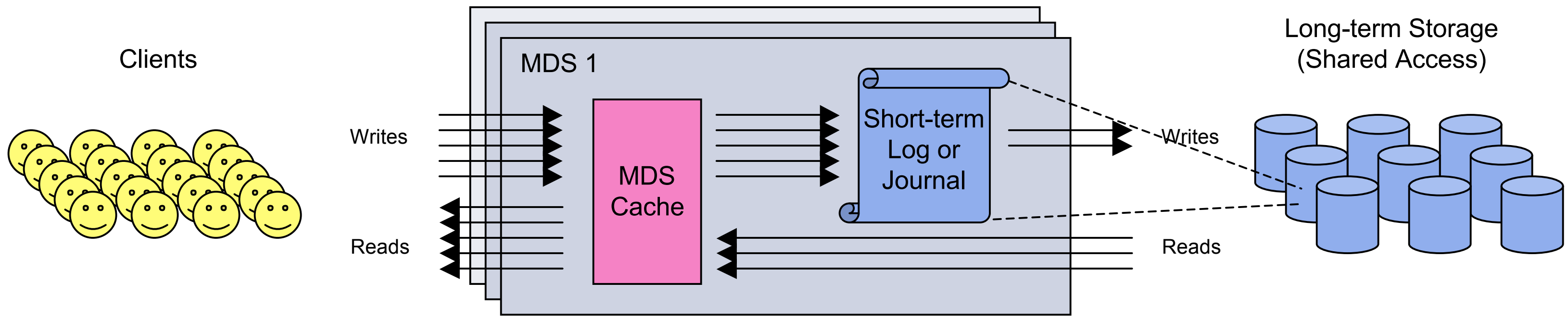


- Consider MDS cluster as an intelligent metadata cache
- MDS cluster must serve both read and update transactions
- MDS cache absorbs some fraction of read requests
- All updates immediately committed to stable storage for safety
  - …but most metadata is updated multiple times in a short period!
- Short-term log absorbs multiple updates
  - Large
  - Flushed very lazily
    - Obsolete updates are discarded
    - Valid updates are applied to regular on-disk metadata structures

ceph

PETABYTE SCALE STORAGE

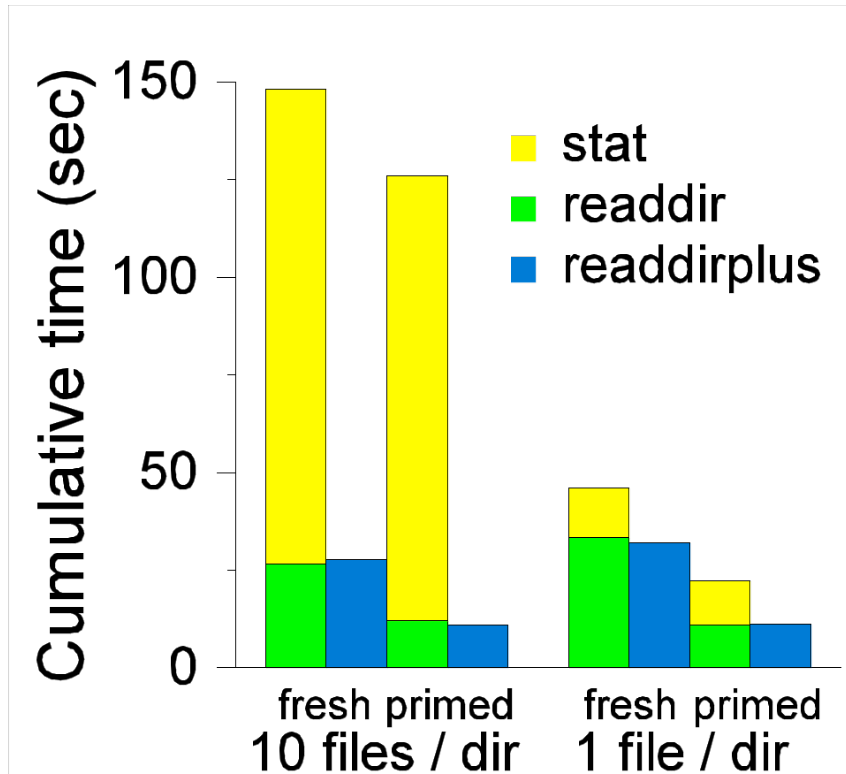
Metadata Storage—Two Tiers



- Short-term storage in metadata journal
  - Updates take advantage of **high sequential write bandwidth**
  - Absorb short-lived or repetitive metadata updates
  - Journal used for recovery after MDS failures
- Long-term storage
  - On-disk layout **optimized for future read access**
    - Group metadata by directory
    - Embed inodes—good locality without large, awkward inode tables

Extending POSIX—readdir(), stat(), and readdirplus()

- Cumulative time for stat() and readdir() or readdirplus() while walking a large directory hierarchy
  - Directory size does not effect readdir() or readdirplus() time
  - readdirplus() (or relaxed consistency) eliminates MDS interaction for obtaining stat() results

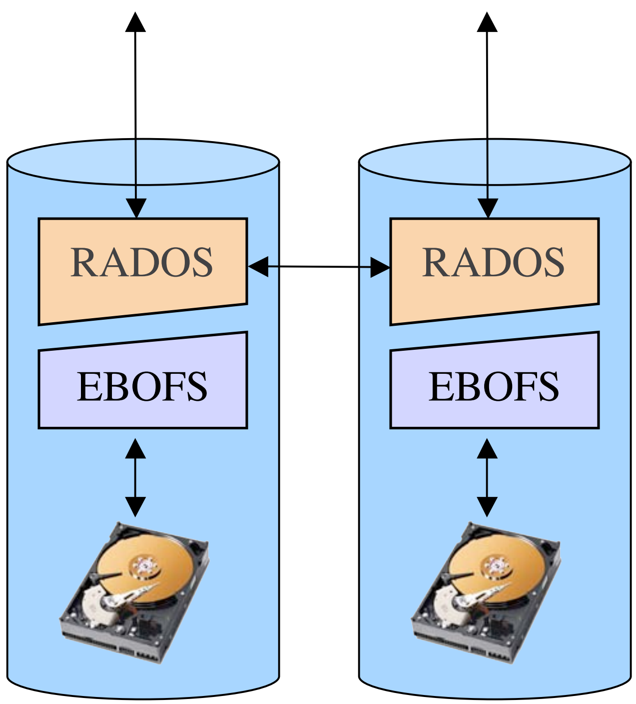


Outline

- Maximal separation of data and metadata
  - Object-based storage
  - Independent metadata management
  - CRUSH – data distribution function
- Dynamic metadata management
  - Adaptive and scalable
- Intelligent disks
  - Reliable Autonomic Distributed Object Store

RADOS—Reliable Autonomic Distributed Object Store

- Ceph OSDs are *intelligent*
  - Conventional drives only respond to commands
  - OSDs communicate and collaborate with their peers
- CRUSH allows us to delegate
  - data replication
  - failure detection
  - failure recovery
  - data migration
- OSDs collectively form a single logical object store
  - Reliable
  - Self-managing (autonomic)
  - Distributed
- RADOS manages peer and client interaction
- EBOFS manages local object storage



RADOS – Cluster map

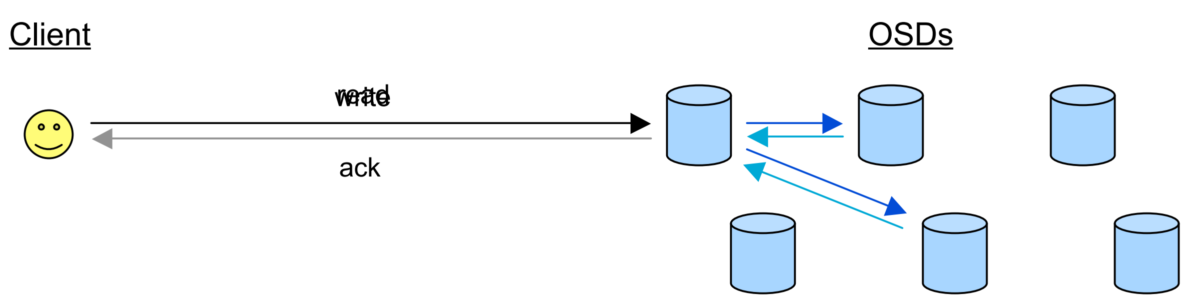
- OSD *cluster map* specifies
  - What OSDs comprise the cluster
  - The CRUSH function mapping each PG to a list of OSDs
- Globally known by all parties (clients, OSDs, MDSs)
  - Object locations are then calculated when needed
- Small “monitor” cluster manages master copy
  - Makes updates when needed

➢ Map allow OSDs to act intelligently and independently

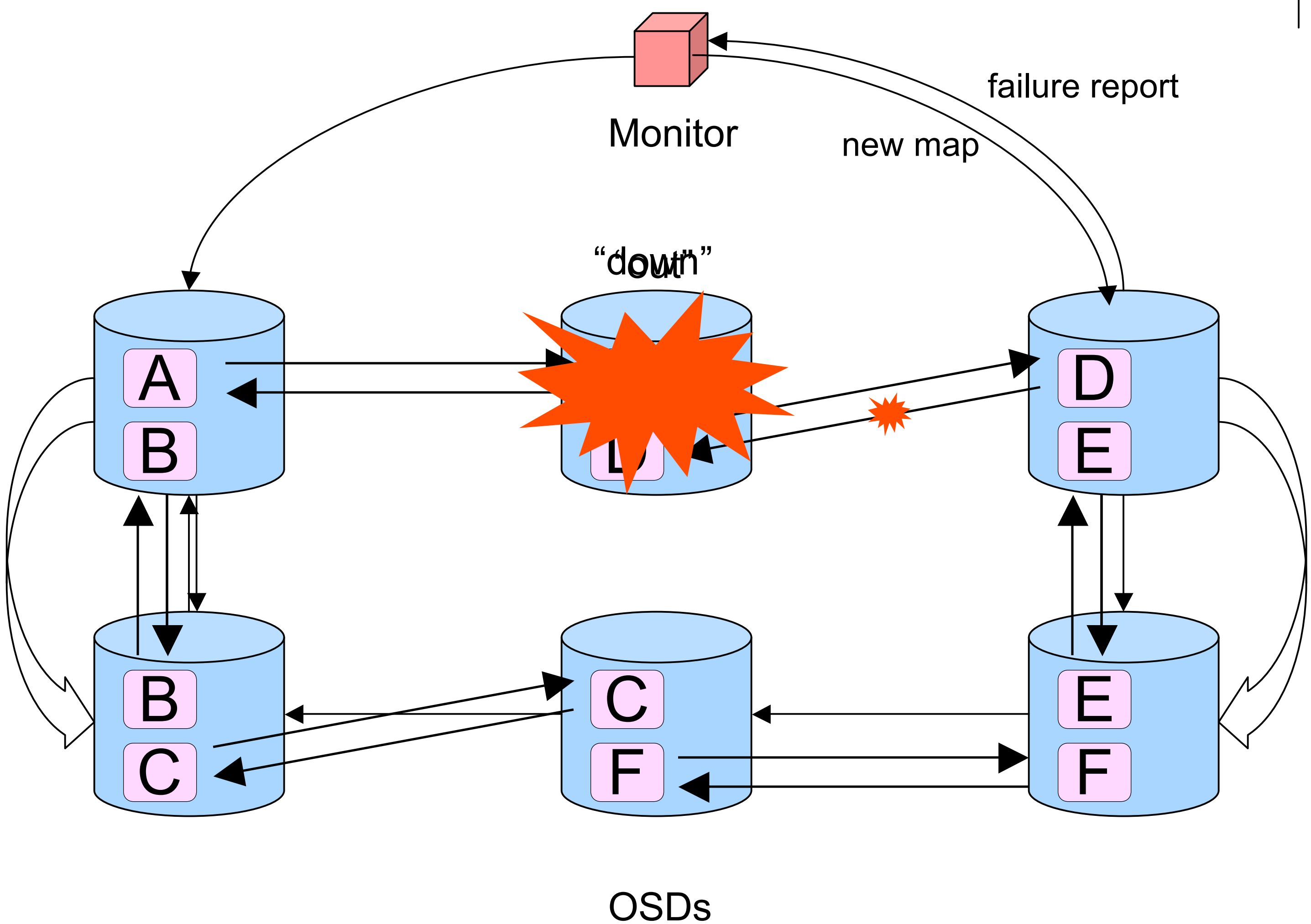
ceph
PETABYTE SCALE STORAGE

RADOS – Data Replication

- Each object belongs to a PG
- Each PG maps to a list of OSDs
- Clients interact with the first OSD (“primary”)
  - Reads are satisfied by the primary
  - Writes are forwarded by the primary to all replicas
    - Leverage local OSD interconnect bandwidth
    - Simplifies client protocol, replica consistency
    - Low incremental cost for replication levels > 2



RADOS—
Failure Detection and Recovery



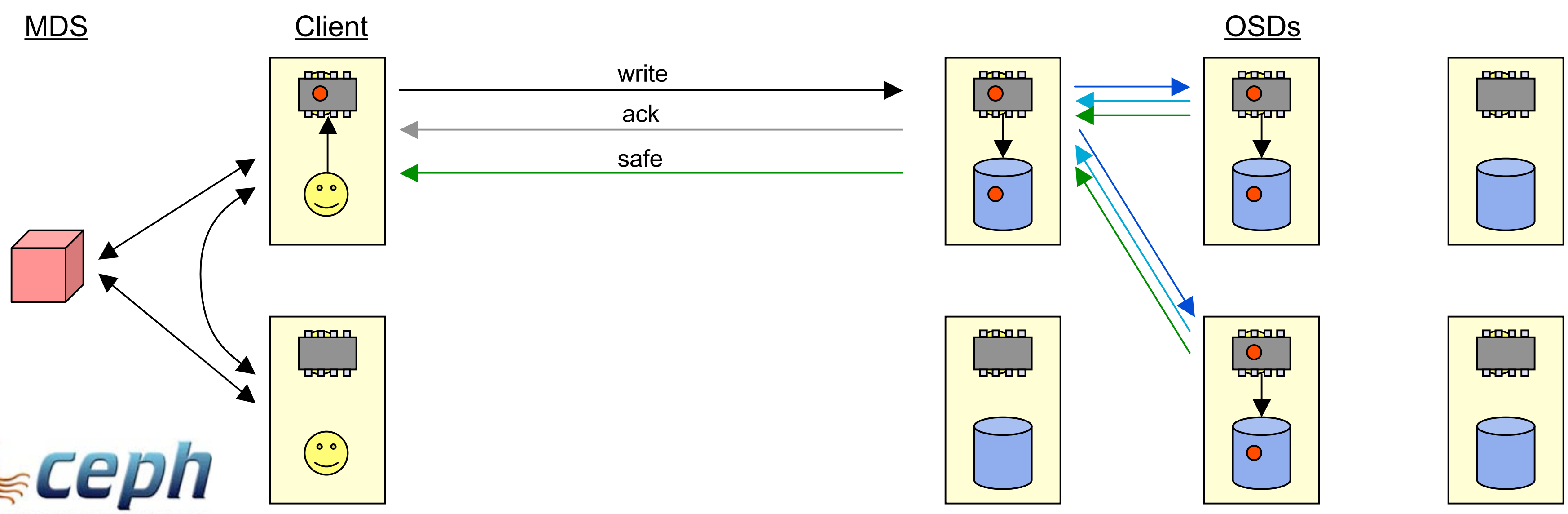
RADOS – Scalability

- Failure detection and recovery are distributed
  - Centralized monitors used *only* to update map
- Maps updates are propagated by OSDs themselves
  - No monitor broadcast necessary

- Identical "recovery" procedure used to respond to all map updates
  - OSD failure
  - Cluster expansion
- OSDs always collaborate to realize the newly specified data distribution

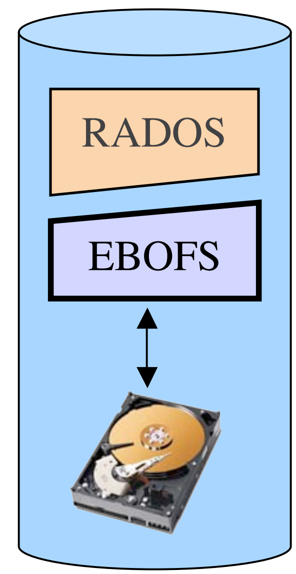
RADOS – Data Safety

- Two reasons we write data to a file system
  - **Synchronization** – so others can see it
  - **Safety** – so that data will be durable, survives power failures, etc.
- RADOS separates write acknowledgement into two phases
  - **ack** – write is applied to all replica buffer cache(s)
  - **safe** – all replicas have committed the write to disk

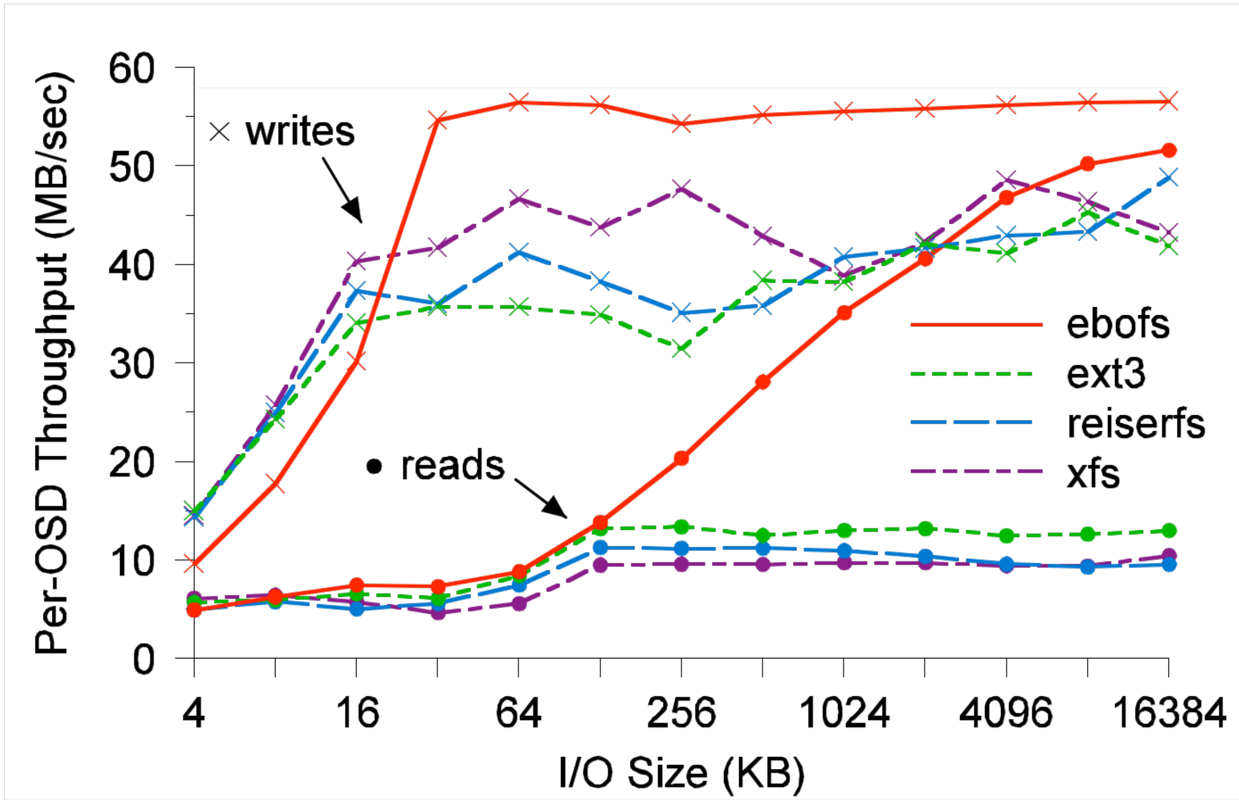


EBOFS—
Low-level object storage

- **E**xtent and **B**-tree-based **O**bject **F**ile **S**ystem
- Non-standard interface and semantics
  - Asynchronous notification of commits to disk
  - Atomic compound data+metadata updates
- Extensive use of copy-on-write
  - Revert to consistent state after failure
- User-space implementation
  - We define our own interface—not limited by ill-suited kernel file system interface
  - Avoid Linux VFS, page cache—designed under different usage assumptions



OSD Performance—
EBOFS vs ext3, ReiserFSv3, XFS



- EBOFS writes saturate disk for request sizes over 32k
- Reads perform significantly better for large write sizes

Conclusions

- Decoupled metadata improves scalability
  - Eliminating allocation lists makes metadata simple
  - MDS stays out of I/O path
- Intelligent OSDs
  - Manage replication, failure detection, and recovery
- CRUSH distribution function makes it possible
  - Global knowledge of complete data distribution
  - Data locations *calculated* when needed
- Dynamic metadata management
  - Preserve locality, improve performance
  - Adapt to varying workloads, hot spots
  - Scale

➢ High-performance and reliability with excellent scalability!

Ongoing and Future Work

- Completion of prototype
  - MDS failure recovery
  - Scalable security architecture [Leung, StorageSS ’06]

- Quality of service
- Time travel (snapshots)
- RADOS improvements
  - Dynamic replication of objects based on workload
  - Reliability mechanisms: scrubbing, etc.

Thanks!

http://ceph.sourceforge.net/

<u>Support from</u>

Lawrence Livermore, Los Alamos, and Sandia National Laboratories