Ceph: A Scalable, High-Performance Distributed File System

Simon Song
60162468
Department of EECS

Motivation - Background & Pain Points

- Background
  - The performance of the file system plays a major role when it comes to system design & optimization, and system designers have long sought to improve the performance of the file system.
  - Traditional solutions:
    - **NFS**: provides a straightforward solution where a server exports a file system hierarchy so that **client can map into their local name space**
    - **object-based storage architecture**: an architecture that is based on where hard disks are replaced with intelligent object storage devices(OSDs) so that **clients interact with a metadata server(MDS) to perform metadata operations, which significantly improves the overall scalability.** Such kind of an architecture combines the CPU, network interface, and local cache with an underlying disk of RAID(Redundant Array of Inexpensive Disks)
- Pain Point - Main problem to solve
  - Adopting such model continues to suffer from the **scalability limitations** due to little or no distribution of the metadata workload.
  - **Continued reliance on traditional file system principles** like allocation lists and inode tables
  - The **reluctance to delegate intelligence to the OSDs**

All these pain points further limits the ***scalability and performance***, and ***increases the cost of reliability***

Main Contributions

Proposed **Ceph**, a distributed file system that provides **excellent performance**, **reliability**, and **scalability**.

How it is achieved:

- **Performance** - Ceph **decouples data and metadata operations** by eliminating file allocation tables and replacing them with generating functions.
- **Scalability** - Ceph also utilizes a **dynamic metadata management** that dramatically improves the scalability of the entire system.
- **Reliability** - Ceph also uses a reliable autonomic **distributed object storage**

This allows Ceph to utilize the intelligent attribute in the OSDs to distribute the complexity surrounding data access, update serialization, replication and reliability, failure detection, and recovery.

Technical Details - Main Components of Ceph

- Main components of Ceph?(1st paragraph of Section 2)
  - **Client** - a cluster of instances that exposes a near-POSIX file system interface to a host or process
  - **a cluster of OSDs** - a cluster that collectively stores all the data and metadata
  - **a cluster of metadata server** - a cluster that manages the namespace(file names and directories) while coordinating security, consistency and coherence

POSIX: Portable Operating System Interface. A a family of standards used for maintaining compatibility between operating systems

- Why do we say that Ceph is a 'near' POSIX? (also in the 1st paragraph of Section 2)
  - In order to improve the system performance and better satisfy the needs of applications, it is better to **extend the interface** and **selectively relax the consistency semantics**.

Technical Details - System Overview

- Decoupled Data & Metadata - Performance - (Section 2 → "Decoupled Data and Metadata")
    - **Separation of the file metadata management** from the storage of the file data
        - The File data is maintained in the Object Storage Cluster, while the metadata is maintained in the Metadata Cluster
    - **Elimination of the allocation list entirely**
        - Used a special-purpose data distribution function called **CRUSH(Controlled Replication Under Scalable Hashing)** to assign the file objects to storage devices. Any party is now able to calculate the location of an object of a file rather than maintaining a long per-file block list.
        - This simplifies the design of the system and reduces the metadata workload
- Dynamic Distributed Metadata Management - Scalability - (Section 2 → "Dynamic Distributed Metadata Management")

  The file system metadata operations takes up to half of the typical file system workloads → **we need an efficient metadata management**!

    - Utilization of a novel metadata cluster architecture based on Dynamic Subtree Partitioning to intelligently distribute the responsibility for managing the file system directory.
    - The workload distribution among metadata servers is based entirely on current access patterns → Ceph can effectively utilize the available MDS resources and achieve a **near-linear scaling** in the number of MDSs.

Technical Details - System Overview

- Reliable Autonomic Distributed Object Storage - **Reliability -** (Section 2 → "Reliable Autonomic Distributed Object Storage")

  Large systems are composed of a large number of devices and require dynamic reactions to situations like new storage deployment and destruction, data creation, movement and deletion, etc.

  → **We have to effectively utilize the available resources** and **maintain a desired level of data replication** to ensure the system will work reliably.

  - Ceph is in charge of data migration, replication, failure detection and recovery to the cluster of OSDs
  - OSDs are providing a single logical object store to clients and metadata servers.

  In this way, Ceph will be able to utilize the CPU and the memory effectively on each OSD to provide a reliable and a highly available object storage with linear scaling.

Technical Details - Client Operation

The Ceph client runs on each host executing the application code and exposes a file system interface to applications.

The client code runs entirely in the user space and can be accessed either by linking to it directly or as a mounted file system via FUSE(a user-space file system interface)

In this paper, the paper discussed about the client operation from three aspects as below:

- File I/O and Capabilities
- Client Synchronization
- Namespace Operations

The paper then talks about the two main components of Ceph that makes the system more scalable and reliable:

- Dynamically Distributed Metadata
- Distributed Object Storage

Technical Details - Client Operation

- File I/O and Capabilities

  Ceph needs to deal with I/O requests from a client.

  When a process wants to open a file…? (1st paragraph of Section 3.1)

  1. the client sends a request to the MDS cluster
  2. The MDS traverses the file system and translate the file name into the *file* inode
     a. if the file exists and the access is granted:
        - MDS returns the inode number, file size, and information about the striping strategy used to map file into objects
        - MDS also grants the client a ‘capability’ specifying what kind of operations are permitted(read, write, etc)
  3. Ceph generalizes a range of striping strategies to map the file data onto a sequence of objects. Objects replicas are then assigned to OSDs using the CRUSH.
  4. Using the inode number, layout, and file size, a process(or a user) can name and locate all objects containing the file data and read directly from the OSD cluster instead of maintaining a long, per-file list and looking it up within the list

Technical Details - Client Operation

- Client Synchronization

  Synchronization is required because there will be a mix of readers and writers to a file at the same time.
  When a file is opened by multiple clients with a mix of writers & readers... ? (1st paragraph of Sec 3.2)

  1. MDS will revoke any previously issued read caching and write buffering capabilities, forcing the client I/O for that file to be synchronous. → This means that the read/write operation of each application will be blocked until it is approved by the OSD(think about the concept of ‘lock’ in Java programming)
  2. When there is a large write operation that affects other objects, a client will obtain an exclusive lock on the affected operations given by the corresponding OSDs and immediately submit the write/unlock operations to finish the write operation.

A drawback of Ceph on the Synchronization side...?(2nd & 3rd paragraph of Sec 3.2)

- synchronous I/O can slow down the performance a lot. Depending on different applications, applications that require a high performance will suffer from the synchronization problem.
- To solve this issue, a set of high-performance extensions to the POSIX I/O interfaces are developed to allow the app to manage their own consistency to achieve a high

Technical Details - Client Operation

- Namespace Operations

  Client interaction with the file system, namespace is managed by the MDS(metadata server) cluster.
  Both read and updates operations are synchronously applied by the MDS to ensure properties such as consistency, correct security, and safety.

  In both scenarios where either performance or consistency is needed, Ceph optimizes the most common metadata access scenario such as a ***readdir*** followed by a sequence of ***stat***, a sequence of command that greatly reduces the performance in large directories.
  In Ceph, what a 'readdir' command actually does is … (2nd paragraph of Sec 3.3)

  - A readdir command requires only a single MDS request that fetches the entire directory including the inode contents.
  - If a readdir is immediately followed by one or more stats, the briefly cached information is returned; otherwise it will be discarded.
  - In the meantime, doing such will take a risk of data inconsistency, but this will greatly improve the performance if a high performance is a top one priority.

Technical Details - Dynamically Distributed Metadata

Using dynamically distributed metadata improves the overall scalability, but what are the reasons that we use this kind of method to achieve high scalability? (1st paragraph of Section 4)

- while metadata often takes up to half of the file system workloads, *“metadata operations involve a greater degree of interdependence that makes scalable consistency and coherence management more difficult”*

Because of this reason, we need to find out a way to effectively manage the metadata directory in order to optimize the metadata operations.

As we have mentioned earlier, Ceph uses a generation function called CRUSH to minimize the metadata workload. What are the methods that the paper uses to minimize the metadata Disk I/O and maximize locality & cache efficiency?(last paragraph of section 4)

- Used **a two-tiered storage strategy** to minimize the metadata related disk I/O
- Used “**Dynamic Subtree Partitioning**” to maximize the locality and cache efficiency

Technical Details - Dynamically Distributed Metadata

Metadata Storage

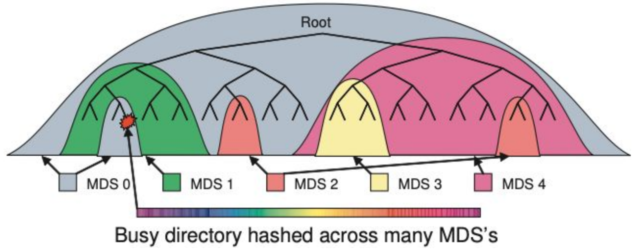
In Metadata Storage, the MDS satisfies most of the requests from its in-memory cache, we still need to commit the changes of the metadata into disk for safety reasons.

In such scenario, what the paper was doing is… (1st paragraph of Sec 4.1)

Introduced a set of large, bounded, lazily flushed “journals”

- Such journals allow each MDS to quickly stream the updated metadata to the OSD cluster in an efficient and distributed manner.
- The journals also absorbs repetitive metadata updates to have a vastly reduced re-write workload

Technical Details - Dynamically Distributed Metadata



Dynamic Subtree Partitioning

Current methods of maintaining the file systems are:

- Employ some form of static subtree-based partitioning → the most existing method
- Use hash functions to distribute directory and file metadata → recent experimental method

However, the limitations of the two ways are...(1st paragraph of Sec 4.2)

- **static subtree partitioning** fails to cope with dynamic workloads and datasets
- **the hashing method** destroys metadata locality and critical opportunities for efficient metadata prefetching and storage

In Ceph, the way we are handling the MDS cluster is... (2nd paragraph of Sec 4.2)

- Use a **dynamic subtree partitioning strategy** that adaptively distributes cached metadata hierarchically across a set of nodes.
- Use a **counter with an exponential time delay** to indicate the popularity of the subtree. → MDS will have a weighted tree structure to keep the workload evenly distributed

Technical Details - Dynamically Distributed Metadata

Traffic Control

Partitioning could provide an even distribution, but it still has some problems when there are a large number of hot spots or flash crowds(many read/ many write).

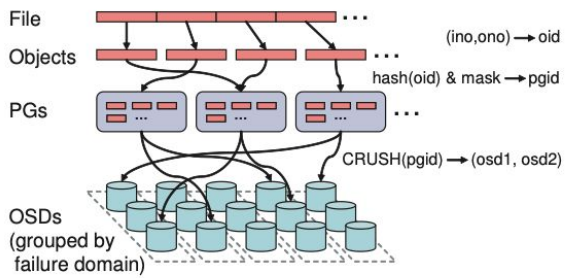
The way that we deal with a large number of hot spots is... (1st paragraph of Sec 4.3)

- For a large number of read:
  The contents of heavily read directories (*e. g.*, many opens) are selectively replicated across multiple nodes to distribute load

- For a large number of write(file creation):
  Directories that are particularly large or experiencing a heavy write workload (*e. g.*, many file creations) have their contents hashed by file name across the cluster, achieving a balanced distribution at the expense of directory locality

This guarantees the MDS of Ceph to be effective under both lightweight partition and heavyweight partitions.

Technical Details - Distributed Object Storage



Data Distribution with CRUSH

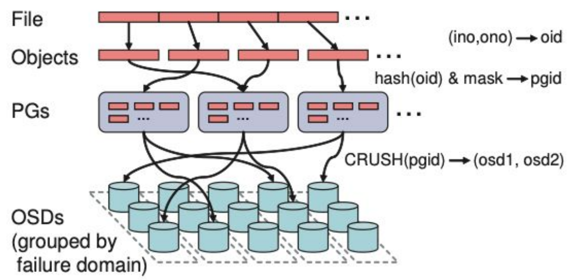
In general, the Ceph clients and metadata servers will see the OSD as a single logical object store and namespace, and there will be normally hundreds of thousands of OSDs. In this case, we have to deal with how the data should be distributed across different OSDs.

Using CRUSH, what the paper does to achieve the data distribution is … (1st paragraph of Sec 5.1)

Data Distribution: Used a stochastic approach to evenly distribute the data…

1. When there is a new device adding in, we select a random subset of the existing OSDs and copy it into the new device.
2. When there is a device being deleted, we randomly distribute all the data within that device to all the other existing devices.

Technical Details - Distributed Object Storage



Data Distribution with CRUSH

Data location:

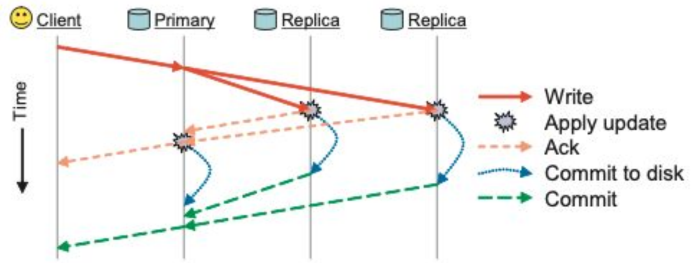
1. Ceph first maps objects into placement groups(PGs) using a simple hash function
2. Choose a value with the amount of replication-related metadata maintained by each OSD to balance the variance in OSD utilizations
3. Assign the placement groups to OSDs using CRUSH
4. To locate any object, CRUSH requires only the placement group and an OSD cluster map: a compact, hierarchical description of the devices comprising the storage cluster.

Two advantages of doing with CRUSH:

1. This approach is completely distributed → Any party(client, OSD, or MDS) can independently calculate the location of any object
2. the map is infrequently updated and is virtually eliminating any exchange of distribution-related metadata

This kind of approach simultaneously solves the problems of both “**data distribution**” and “**data location**”.

Technical Details - Distributed Object Storage



Replication

The paper assumes that unlike Lustre, a system that can build reliable OSDs using RAID, systems in a petabyte scale will have failures as a norm and thus system availability and data safety needs to be ensured in a scalable fashion.

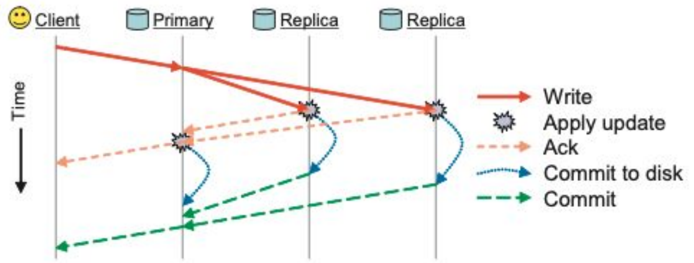
RADOS: Reliable Autonomic Distributed Object Store

What the Ceph will do to handle the data replication is: (2nd paragraph of Sec 5.2)

1. Clients send all the writes to the first non-failed OSD in an object's PG. → A new version number for the object & PG will be assigned, and the write to any additional replica OSDs will be forwarded → update application
2. After all the update in the replicas are ready, the primary OSD applies the update locally and send an acknowledgement to the client.

This greatly reduces the work of the synchronization or serialization between replicas of the client side, and the bandwidth usage is shifted from the client side to the internal OSD network where more bandwidth resources are available.

Technical Details - Distributed Object Storage



Data Safety

The reasons why the data needs to be written in shared storage are… (1st paragraph of Sec 5.3)

- Clients are interested in making their updates visible to other clients.
- Clients are interested in knowing definitively that the data they've written is safely replicated on the disk and will survive over failures such as power outages.

The figure on the top-right corner shows how the data is replicated whenever there is a write request in order to be a safe write:

1. The primary OSD forwards the write to all the other replicas
2. The primary will reply with an acknowledgement after all the OSDs of the in-memory buffer cache.
3. A final commit will be sent to the client when the update is safely committed to the disk.

Technical Details - Distributed Object Storage

Failure Detection

Failure detection is necessary to maintain data safety.

What Ceph does to detect failures is… (1st paragraph of Sec 5.4)

- For certain failures such as disk errors or corrupted data: OSDs can **self report**
- For certain failures that make OSDs unreachable on the network, we will require an **active monitoring** using RADOS.
  - each OSD monitors its peers within the same PG
  - A ping is sent to the peer if an OSD does not hear from the peer for a while
  - two dimensions of the liveness of an OSD are ‘reachable’ and ‘assigned data by CRUSH’

Technical Details - Distributed Object Storage

Recovery and Cluster Updates

The OSD cluster map will change due to OSD failures, recoveries, and the deployment of new storage.

What the OSD does to achieve a fast recovery is ... (1st & 2nd paragraph of Sec 5.5)
OSDs maintain a version number for each object and a log file of all the recent changes for each PG.
To be specific:

- When the cluster map has been updated to a new version, an active OSD will receive the map and re-calculate the CRUSH function to determine the new mapping state within the PG.
- The OSD also has to ‘peer’ with other OSDs within the same PG if the PG’s membership has changed.
- If there are replicated PGs, the OSD must provide the primary with its current PG version number.
    - ...
- The primary then sends each replica an incremental log update such that all parties know what the PG contents should be even if the local object may not match.

Each failure recovery is driven entirely by individual OSDs→ affected PG will be recovered in parallel→ a

Technical Details - Distributed Object Storage

Object Storage with EBOFS

Each OSD manages its local object storage with EBOFS(Extent and B-tree based Object File System).
Implementing EBOFS entirely in user space and interacting with a raw block device allows us to define our own low-level object storage interface and update semantics.

We are using EBOFS instead of other distributed file systems is because… (the disadvantages of other file systems) → (1st paragraph of Sec 5.6)

- The interface and performance are poorly suited for object workloads.
- The existing kernel interface limits our ability to understand when object updates are safely committed on the disk.
- Synchronous writes or journaling provide the desired safety, but only with a heavy latency and performance penalty.
- POSIX interface fails to support atomic data and metadata update transactions

EBOFS provides greater flexibility and easier implementation and avoids cumbersome interaction with Linux VFS and page cache. In the meantime, a user-space scheduler also makes it easier to prioritize workloads or provides quality of service guarantees.

Technical Details - Performance & Scalability Results

Data Performance

The performance of the Ceph prototype is measured under a range of microbenchmarks to test its performance, reliability, and scalability.

Clients, OSDs, and MDSs are user processes running on a Linux cluster and are communicating via TCP.

Measurements of the data performance focus on OSD throughput, write latency, and data distribution and scalability, while the measurements on the metadata performance focus on metadata update latency, metadata read latency, and metadata scaling.

Implementation Results- Performance & Scalability Results

Data Performance - OSD Throughput

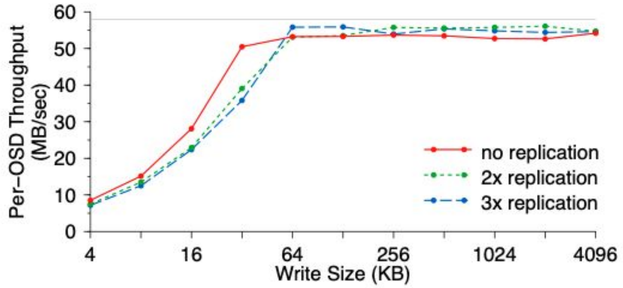


Figure 5: Per-OSD write performance. The horizontal line indicates the upper limit imposed by the physical disk. Replication has minimal impact on OSD throughput, although if the number of OSDs is fixed, n-way replication reduces total *effective* throughput by a factor of n because replicated data must be written to n OSDs.

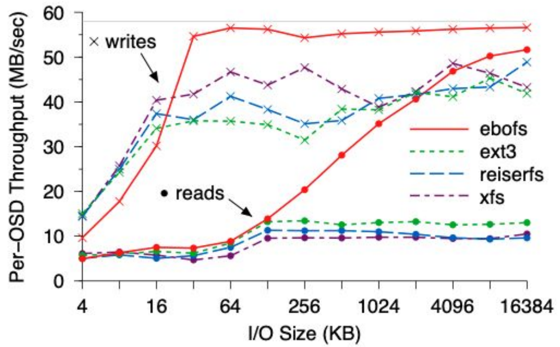


Figure 6: Performance of EBOFS compared to general-purpose file systems. Although small writes suffer from coarse locking in our prototype, EBOFS nearly saturates the disk for writes larger than 32 KB. Since EBOFS lays out data in large extents when it is written in large increments, it has significantly better read performance.

Replication doubles or triples disk I/O, reducing client data rates accordingly when the number of OSDs is fixed.

Although small read and write performance in EBOFS suffers from coarse threading and locking, EBOFS very nearly saturates the available disk bandwidth for writes sizes larger than 32 KB

Implementation Results- Performance & Scalability Results

Data Performance - Write Latency & Data Distribution Scalability

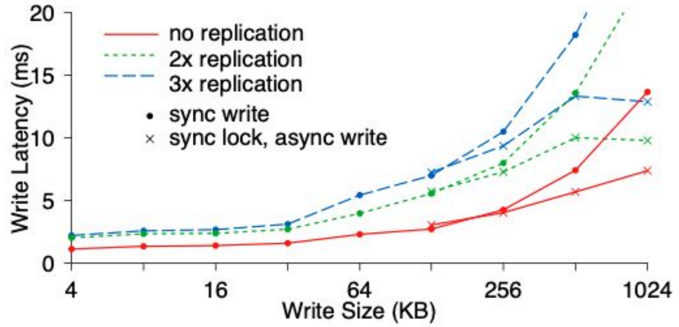


Figure 7: Write latency for varying write sizes and replication. More than two replicas incurs minimal additional cost for small writes because replicated updates occur concurrently. For large synchronous writes, transmission times dominate. Clients partially mask that latency for writes over 128 KB by acquiring exclusive locks and asynchronously flushing the data.

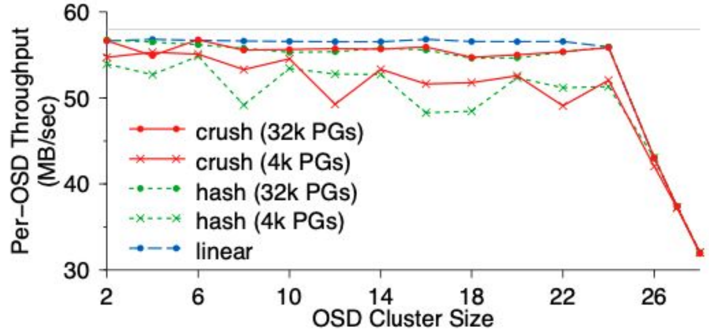


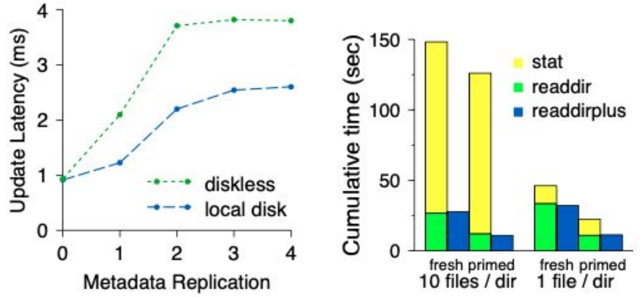
Figure 8: OSD write performance scales linearly with the size of the OSD cluster until the switch is saturated at 24 OSDs. CRUSH and hash performance improves when more PGs lower variance in OSD utilization.

Write Latency

Data Distribution Scalability

Implementation Results- Performance & Scalability Results

Metadata Performance - Metadata Update/Write Latency



(a) Metadata update latency for an MDS with and without a local disk. Zero corresponds to no journaling.

(b) Cumulative time consumed during a file system walk.

Figure 9: Using a local disk lowers the write latency by avoiding the initial network round-trip. Reads benefit from caching, while *readdirplus* or relaxed consistency eliminate MDS interaction for *stat*s following *readdir*.

Metadata Update/Write Latency

Implementation Results- Performance & Scalability Results

Metadata Performance - Metadata Scaling

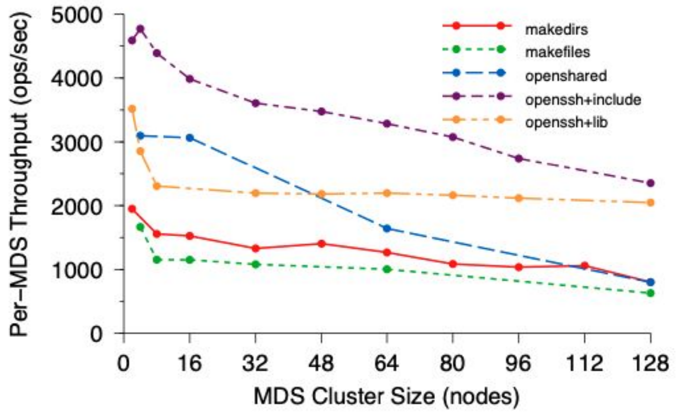


Figure 10: Per-MDS throughput under a variety of work-loads and cluster sizes. As the cluster grows to 128 nodes, efficiency drops no more than 50% below perfect linear (horizontal) scaling for most workloads, allowing vastly improved performance over existing systems.

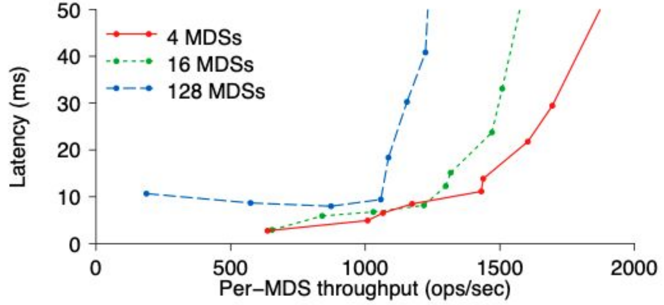


Figure 11: Average latency versus per-MDS throughput for different cluster sizes (*makedirs* workload).

Metadata Scaling

Conclusion

- Ceph addresses three critical challenges of storage systems—scalability, performance, and reliability—by occupying a unique point in the design space
- we maximally separate data from metadata management, allowing them to scale independently. This separation relies on CRUSH
- RADOS leverages intelligent OSDs to manage data replication, failure detection and recovery, low-level disk allocation, scheduling, and data migration without encumbering any central server(s).
- Ceph’s dynamic subtree partitioning is a uniquely scalable approach, offering both efficiency and the ability to adapt to varying workloads.

Future Directions

- Some core Ceph elements have not yet been implemented, including MDS failure recovery and several POSIX calls.
- Two security architecture and protocol variants are under consideration, but neither have yet been implemented.
- investigating the practicality of client callbacks on namespace to inode translation metadata has also not been implemented yet

Thank you!