Ceph: A Scalable, High-Performance Distributed File System

Sage A. Wei, Scott A. Brandt etc.

University of California, Santa Cruz

December 27, 2013

1. Motivation

2. Ceph's Structure
   - Clients
   - Metadata Servers cluster (MDS)
   - Object Storage Devices cluster(OSDs)

3. Evaluation
   - Data Performance
   - Metadata Performance

4. Conclusion and Future work

contents

Background and Present situation

Key Point

The performance of file system have proved critical to the overall performance of tremendous APPs.
Since system designers have long sought to enhance the performance and scalability of distributed storage systems (DFS).

Present Solutions:

1. NFS, sever exports a file system hierachy that clients can map into their local namespace, **limited** to C/S model obstacle.
2. OSD storage, Clients interact with metadata server of metadata oper. and OSDs of file I/O, scalability improved, **limited** little or no distribution of metadata workload.

Design Goals

Then Ceph, a scalable and high-performance DFS, is presented under such circumstances. There are two points of creations:

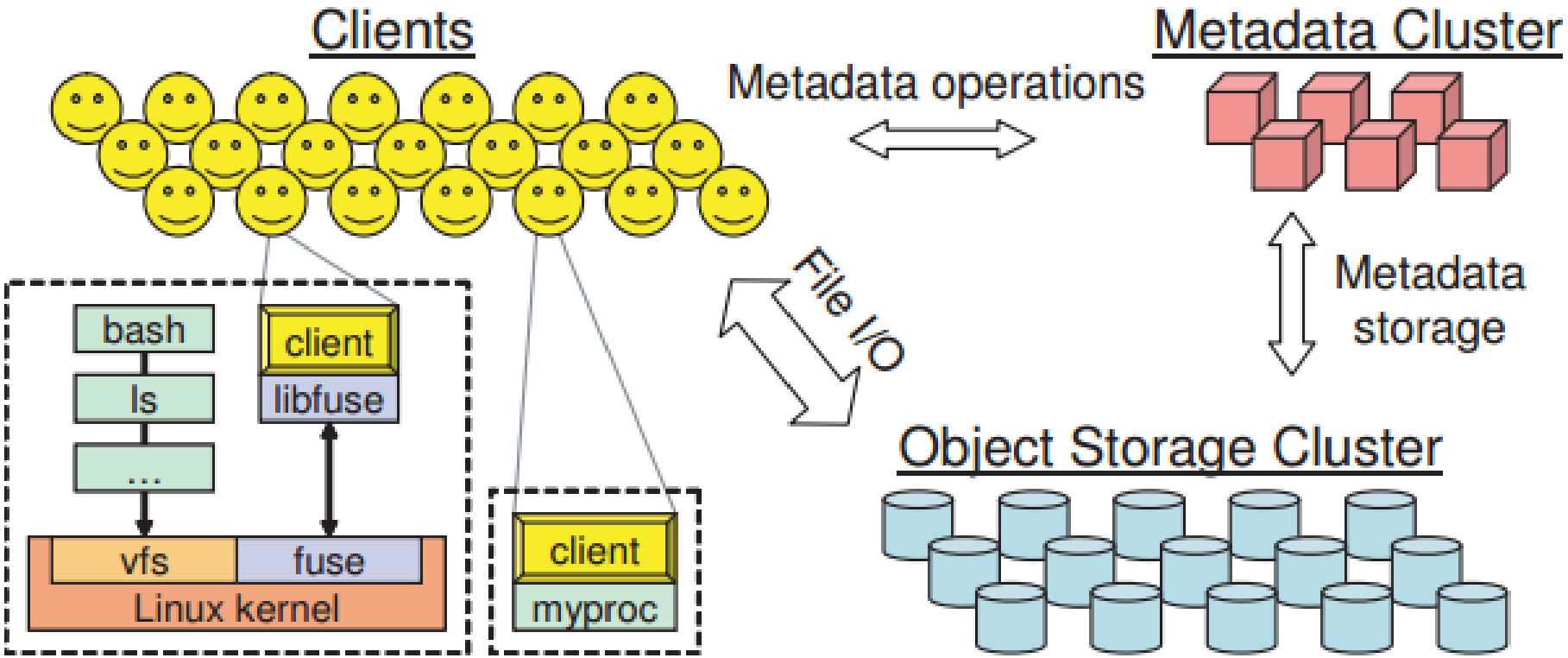
** Decouples data and metadata operations, replacing file allocation tables with Generating functions (CHRUSH).

** A high adaptive distributed metadata cluster architecture, scalability improved.

Assumption: Systems at the petabyte scale are inherently dynamic.

contents

System's Overview



Ceph system has three main components: clients (near-POSIX file system interface), a cluster of OSDs (store all data and metadata), a metadata server cluster(MDS) (namespace, security, consistency and coherence).

File I/O and Capabilities

When a process **opens** a file, the clients **sends** a request to MDS cluster. then MDS **translate** it to the file inode, if it **exists**, the access is granted.

Ceph has lots of striping strategies to **map** file data onto a sequence of objects, and objects names simply combine the **file inode num.** and the **strip number**. And objects replicas are assigned to OSDs using CRUSH (a global mapping function).

Client Synchronization

When a file is **open**ed by multiple clients with either **multiple writers** or a **mix** of readers and writers, MDS **revoke** any previously read caching and write buffer, **force** client I/O be synchronous.

Each APPs read or write **operations** will **block** until is acknowledged by the OSD, making the burden **serialization**, and objects are also to ultilize for **large writes** by acquring **locks**.

Indeed, synchronous I/O can be a performance **killer** for APPs, So it is often desirable to **relax consistency** sometimes, and Ceph supports such relaxation via a **global switch**.

What's more Ceph has acheived a set of POSIX I/O interface, eg. O_LAZY etc.

Namespace Operations

Clients interact with the DFS **namespace** by the **MDS**. And Ceph **optimizes** for the most common metadata access scenarios, **eg**. $readdir$ followed by a $stat$ of each file ...

Ceph could allow consistency to be further **relaxed** by caching metadata **longer**, like the NFS for 30s.

Issues stated above may **break** coherency, so Ceph addresses such problem by revoking any write capabilities to momentarily **stop** from all writers, the highest values are returned with the $stat$ reply.

Although the stop is dramastic, but it is **essential to serialization**.

About the Metadata

A Fact

Metadata oper. often makes up as much as half of file system workloads and lie in the critical path, making the MDS cluster critical to overall performance.

File and directory metadata in Ceph is very small, object names constructed using inode number, and distributed to OSDs using CRUSH, which has **advantages**.

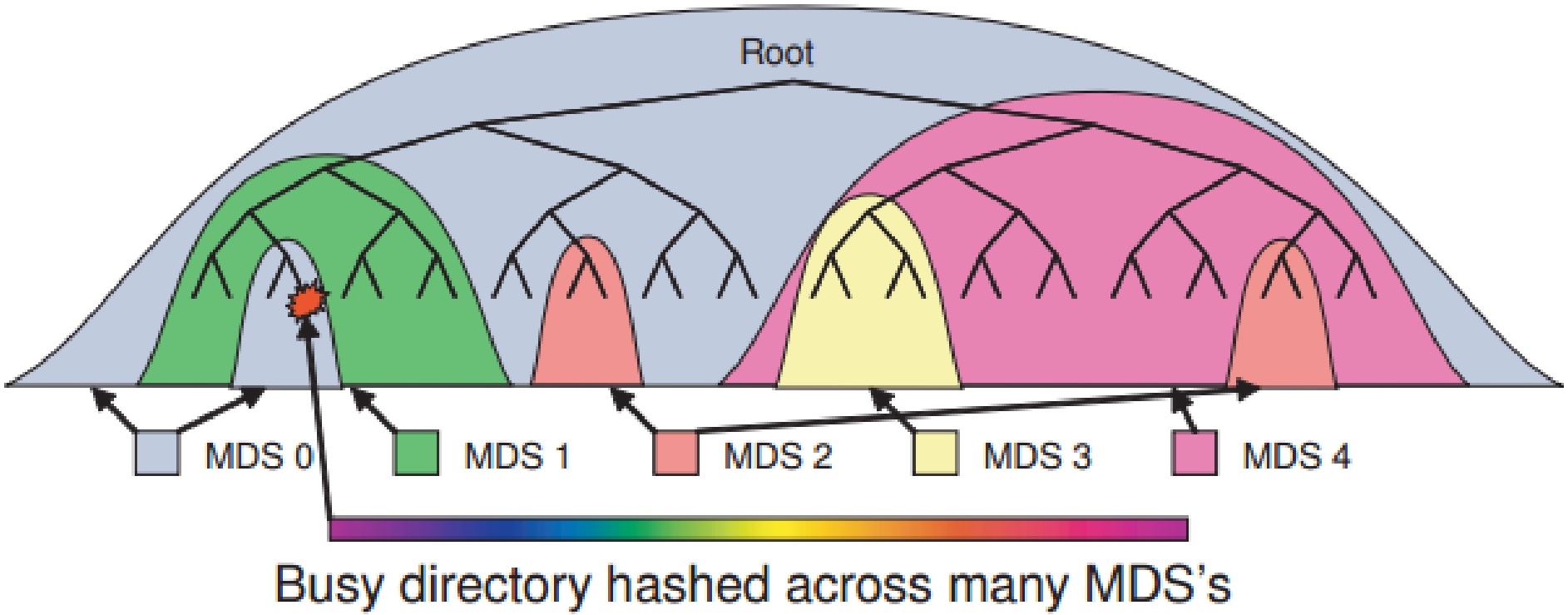
Metadata Storage

A set of large, bounded, lazily flushed **journals** allows each MDS to quickly **stream** its updated metadata to the OSD cluster in an efficient and distributed **manner**.

Such strategy provides streaming updates to disk in an **efficient** fashion, and a vastly **reduced** re-write workloads.

Besides, inodes are embeded within directories, allowing the MDSs to **prefetch** entire directories with a **single** OSD read request.

Dynamic Subtree Partitioning



Background (Static subtree partition and harsh table), then Ceph is based on a dynamic subtree partition strategy.

Ceph dynamically **maps** subtrees of the directory hierachy to metadata servers **based on** the current workload.

Dynamic Subtree Partitioning

Each MDS measures the **popularity** of metadata using counters with time decay. Any **operations increments** the counter on the affected inode and all its ancestors up to the directory.

And each MDS has a weighted tree describing the recent workload distribution, and appropriated-sized subtrees are **migrated** to keep the workload evenly distributed (sharing namespace locks).

Traffic Control

Such strategy can balance a broad range of workload, but can not always cope with **hot spots** or **flash crowds**. So Ceph uses its knowledge of metadata popularity to provide a **wide distribution** for hot spots.

* **read** directories are replicated across multiple nodes to distribute load.
* **write** workload have their contents hashed by file name across the cluster, a balanced distribution while locality expense.

Every MDS **response** provides the client with updated information, allowing them to know what has occurred with their interaction.

overview OSDs

From a high level, Ceph clients and MDS view the OSDs cluster as a **single** logical object store and namespace.

Ceph's Reliable Autonomic Distributed Object Store(RADOS) acheives linear scaling in both capacity and aggregate performance by management of **object replication**, **cluster expansion**, **failure detection**, and **recovery to OSDs**.

Data Distributed with CRUSH

Now Ceph must distribute data among a cluster of thousands of storage devices so that device storage and bandwidth resouces are effectively ultilized, but there comes a problem, **how to avoid imbalance?**

Ceph's **strategy**:

- firstly, map objects into placement groups(PGs) using simple hash function.
- then, PGs are assigned to OSDs using CRUSH(controlled replication under scalable hashing . . .).

Data Distributed with CRUSH

To **locate** any object, CRUSH requires only the PG and an OSD cluster map.
such approach has two key **advantages**:

1. completed distributed, then any party can independently caculate the locatin of any object;
2. map infrequently updated, eliminating any exchange of distribution-related metadata.

CRUSH maps PGs onto OSDs based on **placement rules**, which define the level of replication and any constraints on placement.

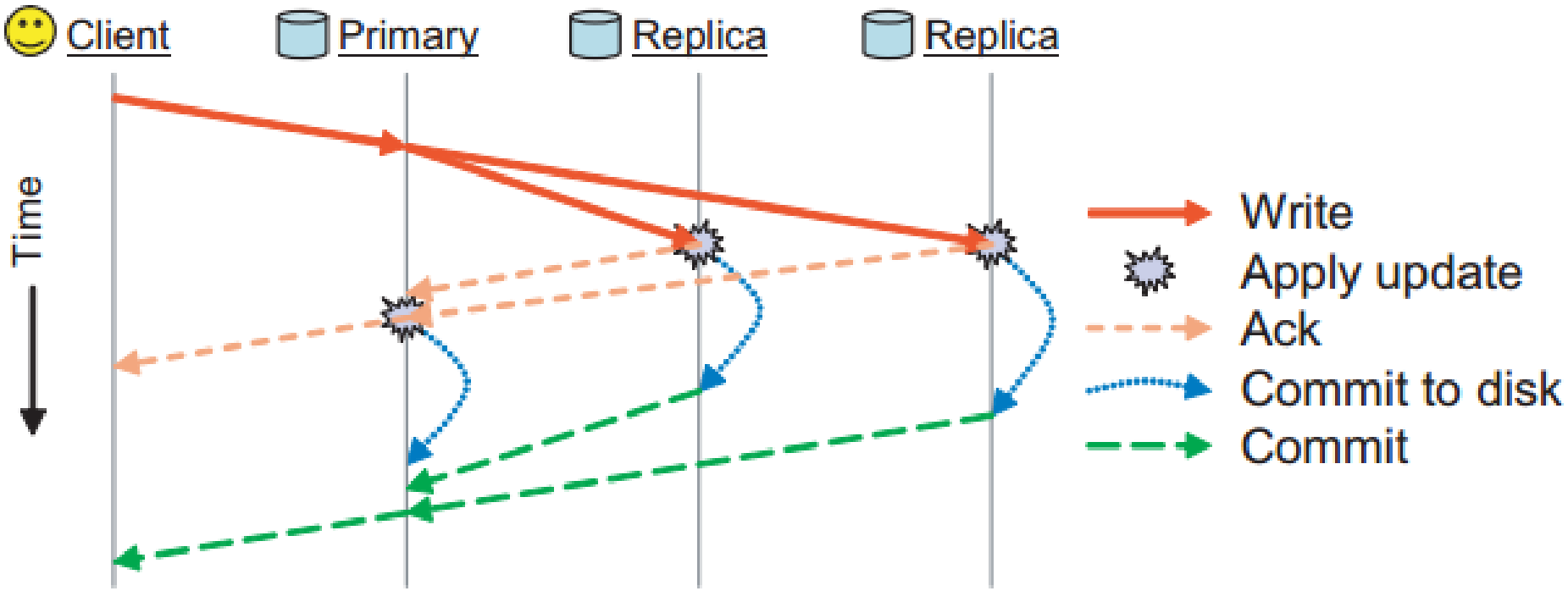
Data Safety

In DFS, there are two reasons why data is written to **shared storage**:

* clients are interested in making their updates visible to other clients;
* clients are interested in knowing that the data they've written is safely replicated.

Then Ceph realizes both low-latency updates for efficient APPs synchronization and well-defined data safety semantic.

Data Safety



Ceph responds with an ack after the write has been applied to the buffer caches on all OSDs replicating the objects.

By default, clients also **buffer** writes until they commit to avoid data loss in the event of a similtaneous power loss to all OSDs in the PG.

Replication and Failure Detection

Replication:
Data is replicated in terms of PGs, each of which is mapped to an ordered list of n OSDs(for n-way replication).

Failure Detection:
Ceph's **failure** has two types: one is disk error or corrupted data, which OSDs can self-report; the other is an OSD unreachable on the network, which requires active monitoring.

ODS **liveness**: an unresponsible OSD is marked $down$, and if the OSD does not quickly recover, it is marked out.

Recovery and Cluster Updates

OSD failure, recoveries etc. will **change** the OSD cluster, Ceph will change such changes **in the same way**. And so Ceph maintain a **version number** for each object and a **log** of recent changes for each PG.

When an active OSD receives an **updated** cluster map, then

1. iterate over all locally stored PGs and CRUSH mapping to determine the primary and replica.
2. if PG has changed, the OSD peers with the PG's other OSDs.(collect info., verify it, then send to all the OSDs).
3. OSDs are then independently responsible for retrieving missing or outdatad objects from their peers.
4. OSD delays processing any request for a stale or missing object.

contents

The Tests' platform

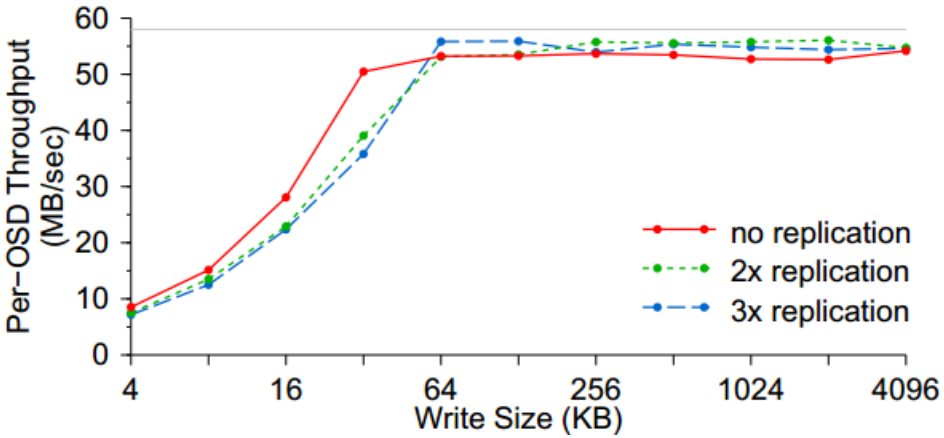
Platform

Clients, OSDs and MDSs are user processes runing on a dual-processor Linux cluster with SCSI disks, TCP communication. OSD or MDS own host, while clients share the same host generating workloads.

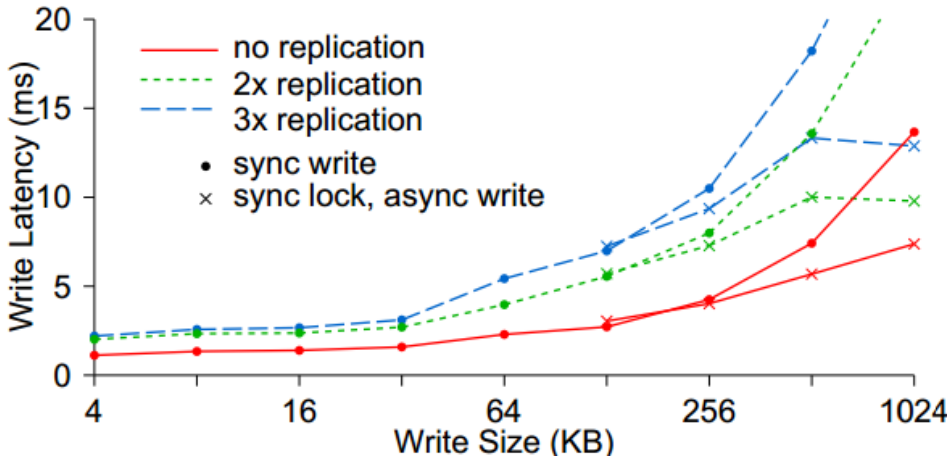
Goals

To demonstrate Ceph's **performance**, **reliability** and **scalability**, focusing on Data and Metadata performance.

OSD Throughput and Write Latency

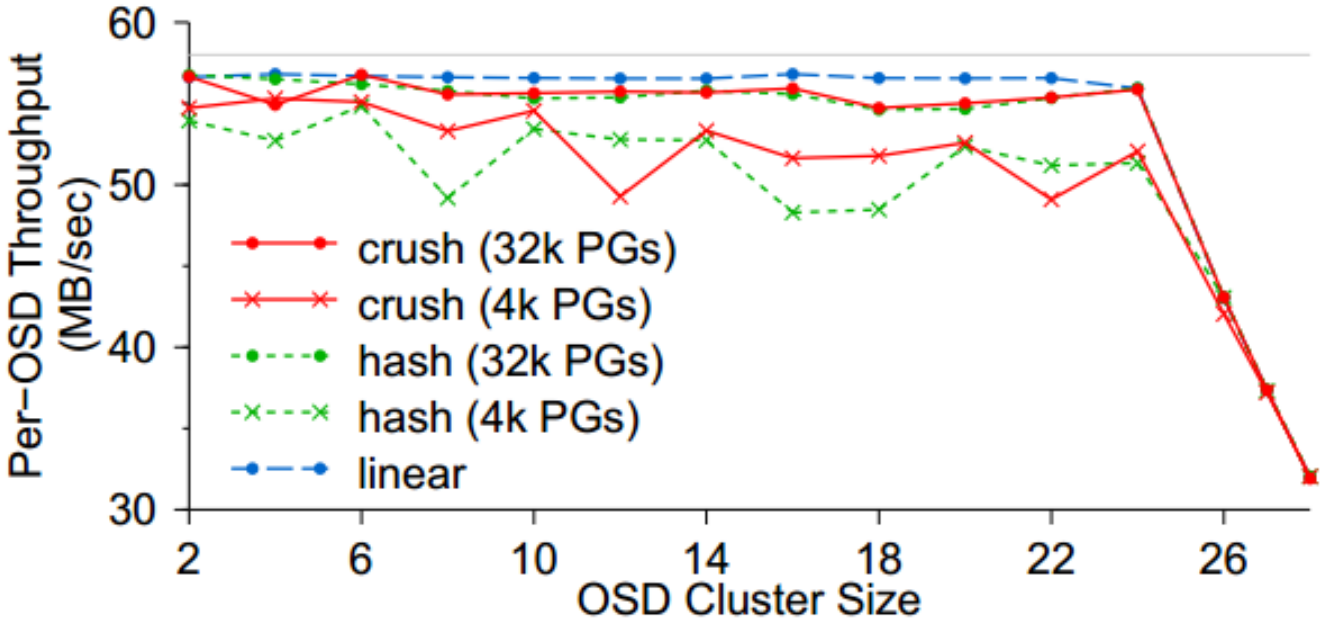


1. horizontal line, upper limit imposed by the physical disk.
2. Replication has minimal impact on OSD throughput.
3. n-way replication reduces total effective throughput due to n.



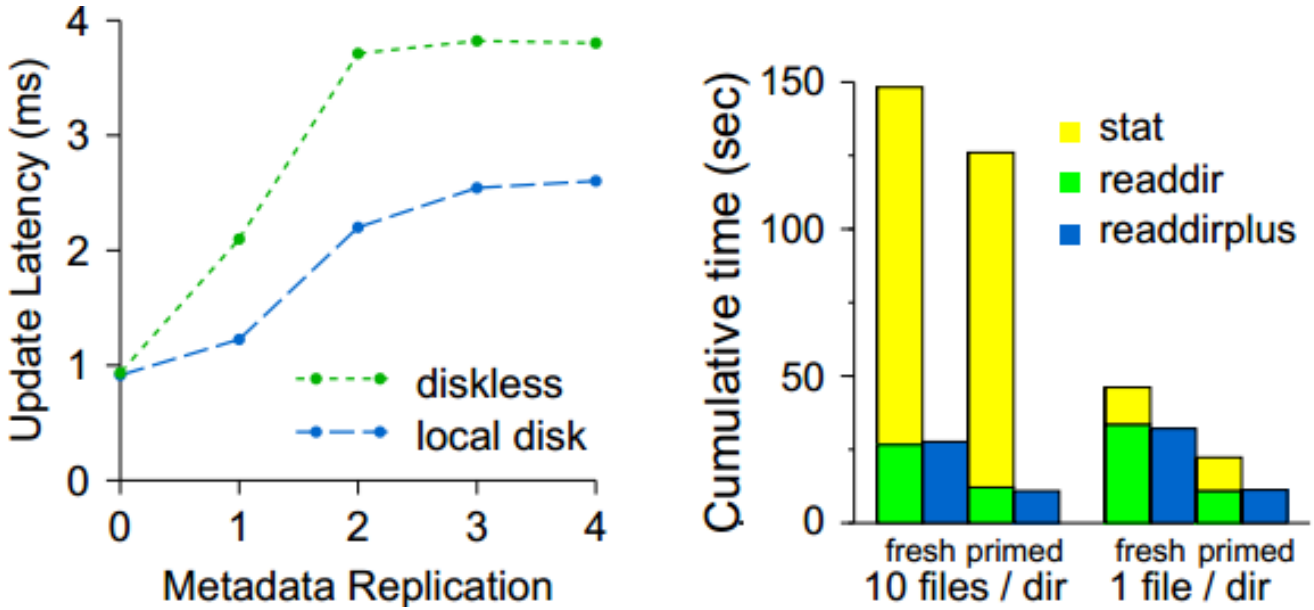
1. small writes minimal additional cost.
2. large synchronous writes, transmission times dominate.
3. Clients mask this latency for writes over 128K by acquring exclusive clocks.

Data Distribution and Scalability



- OSD write performance scales linearly with the size of the OSD cluster.
- the saturated point is at 24 OSDs.
- more PGs and lower variance in OSD utilization can ease such situation.

Metadata Update and Read Latency

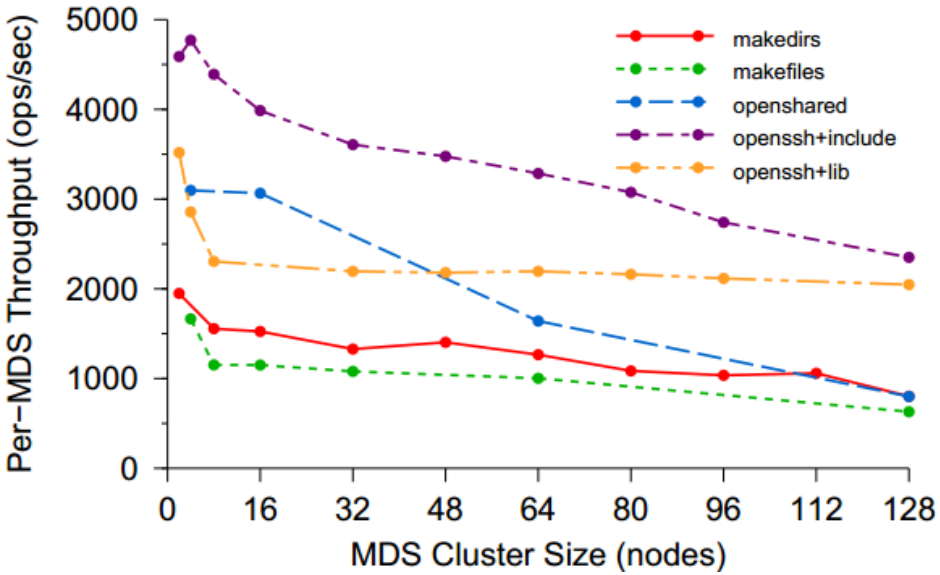


left one: Metadata update latency, and the right one: the cumulative time.

* Using a local disk lowers the write latency by avoiding the initial network round-trip.

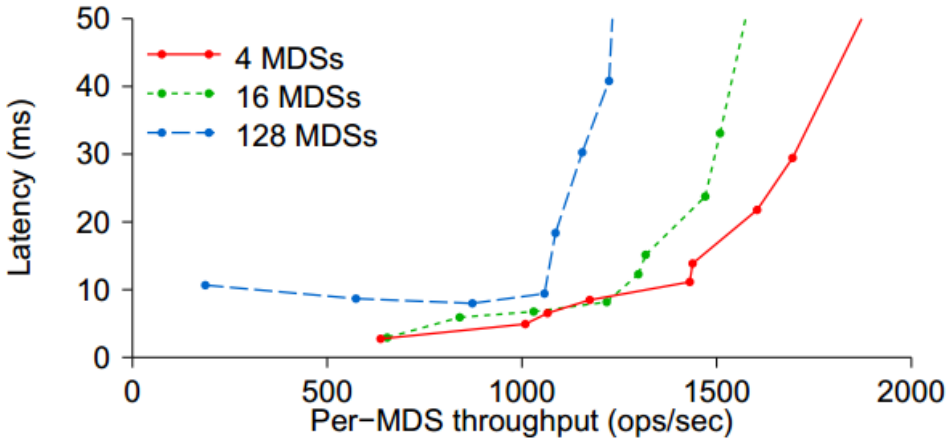
** Reads benefit from caching, and $readdirplus$ eliminate MDS interaction for $stats$ following $readdir$.

OSD Throughput and Write Latency



Per-MDS throughput, workloads and cluster sizes.

cluster grows to 128 nodes, efficiency 50% below, and such issue allows vastly improved performance over existing systems.



Average latency, per-MDS throughput and cluster sizes.

Large clusters have imperfect load distribution, resulting in lower average per-MDS throughput and slightly higher latencies.

contents

Conclusion:

- **Maximize** separate data from metadata management,scale **independently**, relies on CHRUSH.
- Ceph leverages **intelligent** OSDs to manage data replication, failure detection and recovery etc., good **performance**.
- Ceph's metadata management **architecture**, highly scalable storage, **dynamic subtree partition**.

Apart from stated above, there are some points needed to be **enhanced**, eg. MDS failure recovery, POSIX calls, MDS's ability to create snapshots of arbitrary subtrees, OSD's dynamic adjust the level of replication based on the workloads, developing a QoS architecture and so on.

References

1 http://ceph.com/

2 http://www.ibm.com/developerworks/linux/library/l-ceph/index.html?ca=dgr-lnxw01CEPHdth-LX

3 http://deltamaster.is-programmer.com/posts/31908.html

4 http://www.alidata.org/archives/1589

5 http://blog.csdn.net/changtao381/article/details/8698935

6 ...

Questions and answers