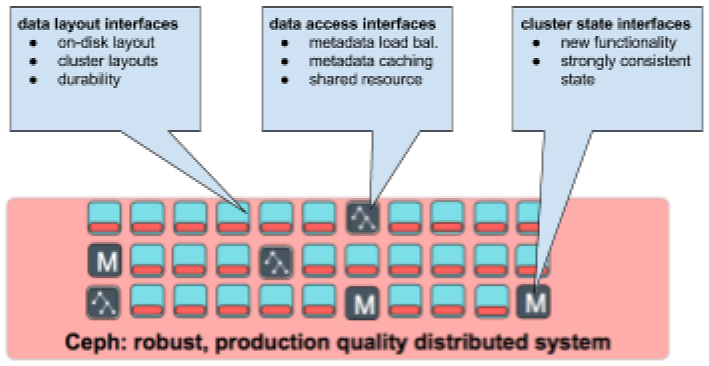
Ceph: An Open-Source Software-Defined Storage Stack

Noah Watkins, Michael Sevilla, Ivo Jimenez, Carlos Maltzahn
University of California, Santa Cruz

Introduction/Theme

High energy physics experiments need a flexible and robust storage stack. Ceph provides storage by striping and replicating data across a reliable object store called RADOS. Ceph can be accessed as a file system (CephFS), a RESTful interface (RADOS Gateway) that supports S3 and Swift, and a block device (RADOS Block Device) commonly used for virtual machine storage. In addition, the underlying object storage system RADOS may be used directly, supporting a rich set of object-based storage abstractions. To provide these APIs Ceph has built many internal abstractions, some of which are exposed to higher level services. **This whitepaper will give a survey of some of the less publicized features of Ceph that can help applications tailor the storage system to their data layout, data access, and cluster availability needs.**

A Ceph cluster is composed of object storage daemons (OSDs), monitor daemons (MONs), and metadata server daemons (MDSs). OSDs provide data fragments; monitors maintain cluster state, and metadata servers (which are optional) manage the POSIX namespace for the Ceph file system layer.



Data Layout, Organization, and Locality

Applications that scale past the limits of existing storage abstractions such as POSIX file systems have begun to adopt alternative storage interfaces to handle both bulk data and metadata management. Over the last decade the predominant I/O interface for new applications seeking to avoid the file abstraction has been the object, which offers high scalability, but relies on applications to handle some aspects of data management and naming.

Many large scale applications use data models that have a natural decomposition onto an object storage system. As developers take advantage of object storage they often times find themselves building domain-specific data access methods on top of objects that provide an opaque bytestream abstractions. This is a powerful approach, but often forces applications to reproduce existing functionality, or use complex protocols like two-phase commit to achieve a desired semantics. The introduction of Ceph and RADOS have changed the landscape by providing an object storage that offers rich data access methods, extensibility, and is open-source, alleviating many concerns with vendor lock-in common with proprietary systems.

To illustrate the power of extensibility in RADOS we highlight a number of projects under way at the Center for Research in Open Source Software (cross.ucsc.edu) at UC Santa Cruz and in collaboration with the Ceph developer community at Red Hat and elsewhere that use a spectrum of Ceph features, which would be challenging to build on other systems.

High-performance Distributed Shared-Log

Distributed logs have been receiving a lot of attention lately. And rightfully so---as a building block, they are a basic concept that in many instances can simplify the construction of distributed systems. But building a distributed log is no simple task. Paxos is the common approach to building a strongly consistent shared log, but it is difficult to scale to high-performance. Our approach is based on the design of the high-throughput CORFU distributed shared log [1]. Briefly, CORFU uses a cluster of flash devices onto which a logical log is striped, spreading I/O load across the cluster. A non-persistent sequencer is used to assign a global order, and the complexities of recovery are handled by re-synchronizing clients when the network counter fails.

We chose Ceph as a platform for building an implementation of the CORFU protocol because Ceph's software-defined capabilities allow us to avoid making assumptions about the media being used. Appends can be directed to high-performance NVMe, and later be moved to slower, cheaper disk or SSD. This flexibility is difficult without the abstractions provided by Ceph.

The core component of Ceph that we rely on is the object class interface that supports the creation of new object interfaces. This is critical for the CORFU protocol in maintaining correctness. Multiple implementations have been build that exploit different characteristics of hardware and low-level data management features, but complexities such as fault-tolerance and replication are never altered.

Lua-based Server-side Processing and Application Interfaces

The RADOS object storage system supports custom interfaces by injecting C++ modules that define new access methods that are expressed as a composition of existing low-level I/O APIs that run in an atomic context. This allows new interfaces like atomic compare and swap, or an interface that keeps an index up-to-date when writes are recorded.

The challenge is that currently there is little flexibility for applications to install new interfaces. We have added a feature that allows new interfaces to be defined using the high-performance scripting language Lua. This gives applications the ability to define new interfaces on the fly.

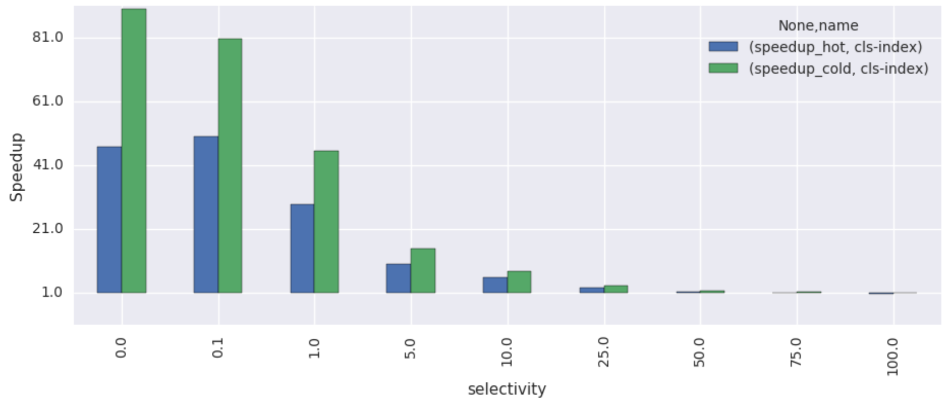
The code snippet below runs on a client which is storing a thumbnail, but later wants to be able to retrieve the thumbnail by size. The "local script" variable defines a script that runs alongside the write remotely and updates a RocksDB index that records the size of the object and the offset within the object where the copy of the object with the desired dimensions can be found.

```
function put_smart(object, filename)
  -- define the script to execute remotely
  local script = [[
  function put(img)
    -- write the input blob
    local size, offset = #img, 0
    cls.write(offset, size, img)

    -- update the leveldb index
    local loc_spec_bl = bufferlist.new()
    local loc_spec = size .. "@" .. offset
    loc_spec_bl:append(spec)
    cls.map_set_val("original", loc_spec_bl)
  end
  cls.register(store)
  ]]

  -- read the input image blob from the file
  local file = io.open(filename, "rb")
  local img = file:read("*all")

  -- remotely execute script with image as input
  clslua.exec(ioctx, object, script, "put", img)
end
```

Remote Data Filtering and Access Methods

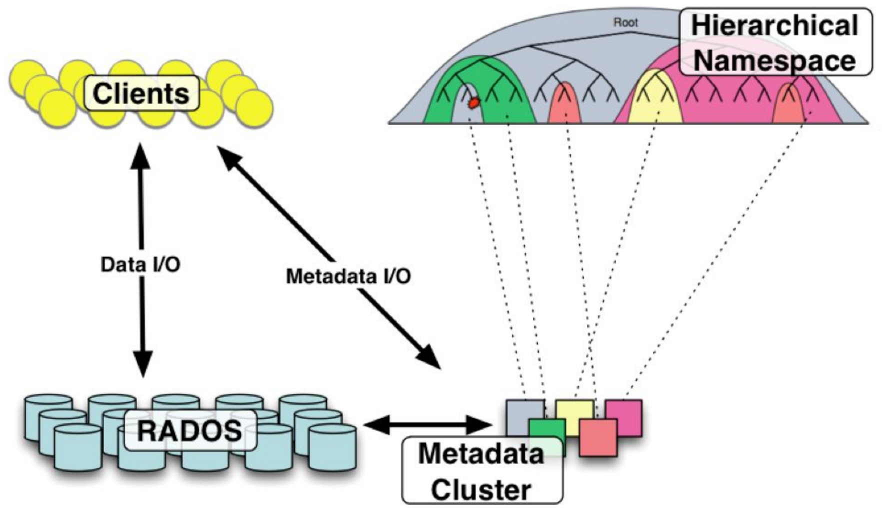
Reducing data movement in data-intensive systems is an important optimization that reduces energy usage, and can have significant impacts on performance. While RDMA and other high-performance networking technologies may reduce this benefit, the same mechanisms that reduce data movement (e.g. locality or data reductions and aggregations) are also useful for exploiting remote resources such as RAM and CPU.

We have shown that we can achieve significant speed up performing remote data reduction via filtering (e.g. database scan with predicate, or computing an isoline). The graph below shows the speedup achieved by server-side filtering over shipping all data to a client. In particular we make use of a server-side index that allows us to track statistics of each object and avoid processing when possible.



Metadata Layout, Access, and Management

File system metadata workloads impose small and frequent requests on the underlying storage. This skewed workload makes it difficult to scale metadata IO in the same way that data read/write throughput scales. CephFS is the POSIX compliant file system that uses RADOS and it has strategies for helping metadata IO scale, such as decoupling metadata/data IO and providing mechanisms for detecting and migrating metadata load to other servers. Although other high energy physics groups are exploring techniques that do not use dedicated metadata servers (e.g., libradosFS [2]), our discussion here focuses on the metadata cluster packaged with CephFS. The architecture of CephFS is shown below:

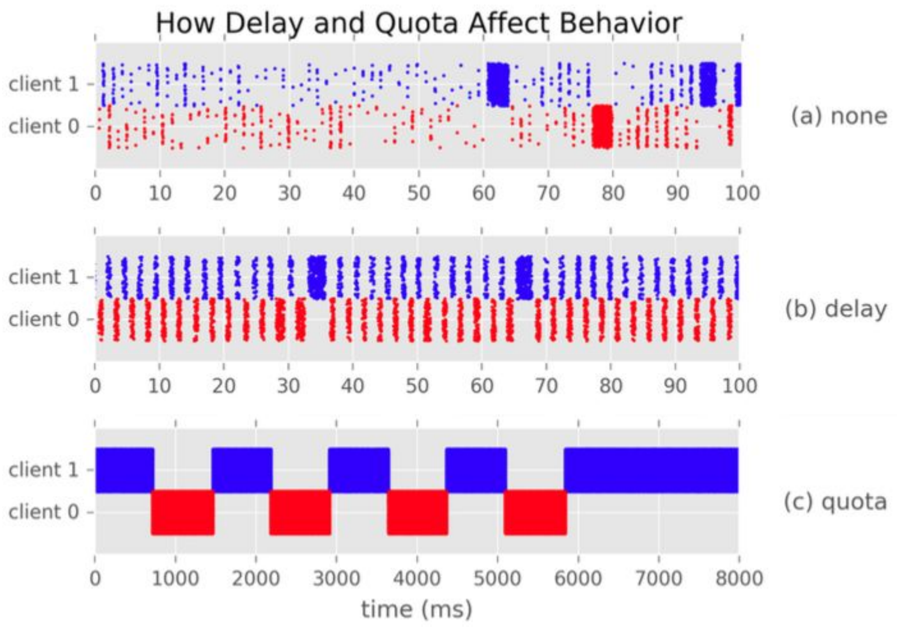


File system metadata is serviced by a separate metadata cluster. This metadata cluster exposes a hierarchical namespace to the clients. The metadata is kept in the collective memory of the metadata cluster and for fault tolerance CephFS (1) streams a journal of updates into RADOS and (2) materializes the namespace as a directory per object in RADOS. To reduce the number of RPCs on a directory listing, file inodes are embedded within directories when they are stored in RADOS. To efficiently serve POSIX file system metadata, the metadata service mediates access to shared resources, balances load across a cluster, and caches hot metadata.

Guarding Shared Resources in the Namespace

Despite some edge cases [4], CephFS is POSIX compliant which means that it provides exclusivity and strong consistency for the files in the hierarchical namespace. This is implemented with capabilities and distributed locking. Each file and directory has an associated state machine so clients can transition to reading, reading and updating, reads caching, writing, buffering writes, changing the file size, and performing lazy IO.

Our previous work added programmability to the capability mechanisms. When multiple clients request the same metadata, the metadata server grants the capability to clients and revokes the capability from clients at a short, hard-coded intervals. We added a time-based delay and quota for the number of requests so we could control latency and throughput. The programmability of the capability grant/revoke intervals are shown below.

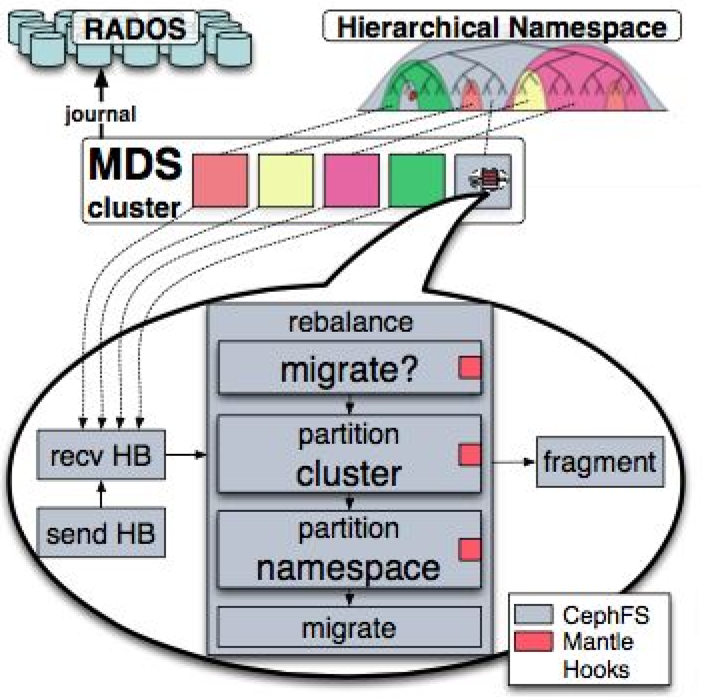


In that figure, the x-axis is time and each dot is a request, spread randomly across the y-axis to show density. The top figure shows the default behaviour for sharing the capability, which is unpredictable. The middle graph is when we control the delay. We see longer bursts of requests and more predictable performance. The bottom graph is when we set a quota (where each client gets the capability for a number of requests) and a timeout. The blocks are much longer and clients do not grab and keep the capability because we see at time 6000 seconds the red client finishes and releases the capability.

Our future work will add more programmability to the capability and data access subsystems so we can replicate and evaluate new techniques for HPC workloads. For capabilities, we want to be able to replicate HPC strategies for reducing synchronization and serialization overheads by transferring responsibilities to the the client. We can do this by allowing the clients to perform metadata transactions locally and delaying the metadata merge. This helps us explore a range of consistency guarantees for namespaces, such as eventual, strong, and implied consistency. For data access, we want to be able to change the way we access and store data with customizable file types [3]; we would store code and policies for accessing data in the metadata structure so that clients know how to read data (e.g., row vs column major scans) or how to interpret specific file layouts (e.g., ROOT or HDF5).

Fault Tolerance and Load Balancing the Namespace

CephFS uses a cluster of metadata servers for both fault tolerance and load balancing. Multiple metadata servers can be configured in active-standby so that backup servers can take over if the active fails. For load balancing the metadata servers can be configured in active-active mode. In this mode, metadata servers manage different subtrees in the namespace using a technique called Dynamic Subtree Partitioning. This approach carves up the file system namespace and distributes the subtrees across the metadata cluster. Large directories are partitioned for write heavy metadata workloads and replicated for read heavy metadata workloads. The balancing logic is shown below:



The decision making is a synchronous process, where every 10 seconds each metadata server makes an independent migration decision. For scalability, there is no global scheduler and each metadata server has an inconsistent view of the namespaces. Each metadata server exchanges heartbeats ("recv HB") and uses the load from other metadata servers to decide whether to migrate ("migrate?"), how to "partition the cluster", how to "partition the namespace", and whether to "fragment" directories into smaller pieces. The administrator can control how load is calculated using CephFS configuration knobs; right now CephFS allows a CPU-based, workload-based, and hybrid balancers.

Our work tries to make the balancing logic more flexible by giving administrators the ability to program new custom logic. Mantle is a programmable metadata load balancer [5]; it provides a general framework and specification for expressing POSIX metadata load balancing policies on the *same storage system*. This lets us compare load balancing strategies instead of the properties of the storage systems themselves and helps future administrators understand the trade-offs of different metadata migration decisions. For example, when() is a callback provided by the API and can be programmed to initiate balancing under different conditions:

```
-- balance when my neighbor is idle
if MDSs[whoami+1]["cpu"]<0.25

-- balance when I have load and my neighbor does not
if MDSs[whoami]["load"]>.01 and MDSs[whoami+1]["load"]<.01 then
```

Caching Metadata and the Namespace

In a sense, the metadata cluster is a large distributed cache that services metadata requests. The full namespace structure and its inodes are stored in RADOS but clients can get metadata from memory by contacting one of the metadata servers. CephFS addresses fault tolerance with a metadata journal that streams into the resilient object store. Similar to LFS and WAFL, the CephFS metadata journal can grow to large sizes ensuring (1) sequential writes into RADOS and (2) the ability for daemons to trim redundant or irrelevant journal entries. A common concern that is that the cost of streaming a journal of updates for metadata operations does not overwhelm an the RADOS cluster.

Each metadata server also employs a two-level LRU cache to improve metadata access for hot and cold data. This cache stores inodes and reduces the number of RPCs between clients and metadata servers. If a client has the directory inode cached it can do metadata writes (e.g., create) with a single RPC. If the client is not caching the directory inode then it must do multiple RPCs to the metadata server to (1) determine if the file exists and (2) do the actual create. Unless the client immediately reads all the inodes in the cache, the inode cache is less useful for create-heavy workloads because the cached inodes are unused. CephFS tries to keep the cache at 100 thousand inodes so the degradation in performance indicates that the metadata server cannot keep up with the workload.

Cluster Monitoring

cluster state interfaces

- new functionality: controls the propagation of binaries and interfaces
  → mon interface
- strongly consistent state: controls the propagation of service-specific metadata
  → mon interface

Future Work in Programmability

Tell them what we have in store and prove we have awareness of the problem

- problem is described in the working group white paper [link]

Awareness of the problem
- how we hope to address that with programmability

Programmable storage systems expose internal services and abstractions to facilitate the construction of new services. By re-purposing existing subsystem abstractions of the storage stack, future programmers can re-use, extend, and compose the functionality of existing storage abstractions in higher-level services. We have built a programmable storage system called Malacology on Ceph because it offers a broad spectrum of existing services, including distributed locking and caching services provided by metadata servers, durability and object interfaces provided by the backend object store, and propagation of consistent cluster state provided by the monitoring service.

References

[1] M. Balakrishnan et al., CORFU: a shared log design for flash clusters. Conference on Networked Systems Design and Implementation (NSDI'12).
[2] J. Rocha, Announcing libradosfs, http://www.joaquimrocha.com/2015/09/01/announcing-libradosfs/.
[3] N. Watkins et al., DataMods: Programmable File System Service. Parallel Data Storage Workshop (PDSW'12).
[4] Ceph documentation, Differences From POSIX, http://docs.ceph.com/docs/master/cephfs/posix/.
[5] M. Sevilla et al., Mantle: A Programmable Metadata Load Balancer for the Ceph File System, Conference on Supercomputing (SC '15).