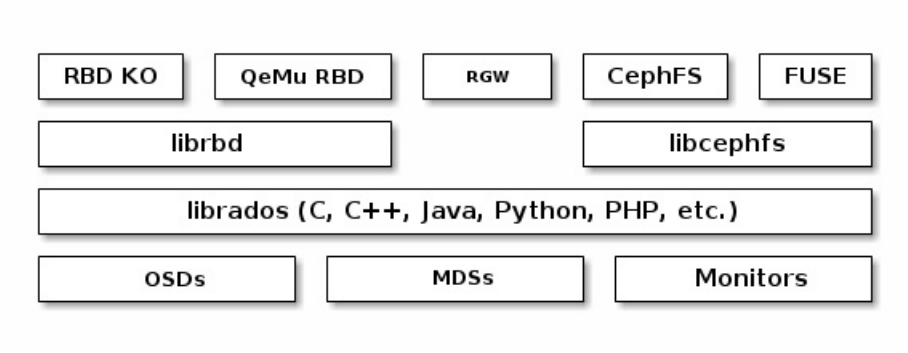
ARCHITECTURE

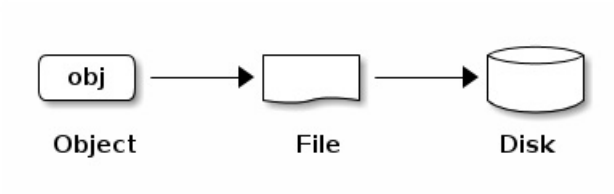
Ceph provides an infinitely scalable Object Store based upon a RADOS, which you can read about in RADOS - A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters. Storage clients and OSDs both use the CRUSH algorithm to efficiently compute information about data location, instead of having to depend on a central lookup table. Ceph's high-level features include providing a native interface to the Object Store via `librados`, and a number of service interfaces built on top of `librados`. These include:

- **Block Devices:** The RADOS Block Device (RBD) service provides resizable, thin-provisioned block devices with snapshotting and cloning. Ceph stripes a block device across the cluster for high performance. Ceph supports both kernel objects (KO) and a QEMU hypervisor that uses `librbd` directly–avoiding the kernel object overhead for virtualized systems.
- **RESTful Gateway:** The RADOS Gateway (RGW) service provides RESTful APIs with interfaces that are compatible with Amazon S3 and OpenStack Swift.
- **Ceph FS**: The Ceph Filesystem (CephFS) service provides a POSIX compliant filesystem usable with `mount` or as a filesytem in user space (FUSE).

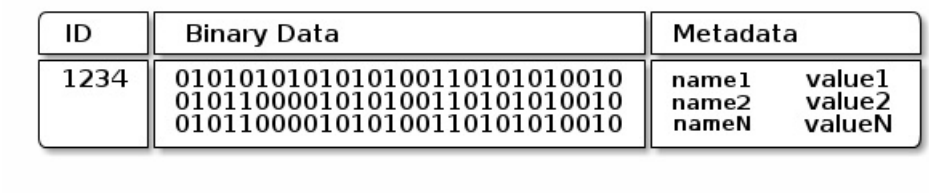
Ceph can run additional instances of OSDs, MDSs, and monitors for scalability and high availability. The following diagram depicts the high-level architecture.



Ceph's Object Store takes data from clients–whether it comes through RBD, RGW, CephFS, or a custom implementation you create using `librados`–and stores them as objects. Each object corresponds to a file in a filesystem, which is typically stored on a single storage disk. `ceph-osd` daemons handle the read/write operations on the storage disks.



OSDs store all data as objects in a flat namespace (e.g., no hierarchy of directories). An object has an identifier, binary data, and metadata consisting of a set of name/value pairs. The semantics are completely up to the client. For example, CephFS uses metadata to store file attributes such as the file owner, created date, last modified date, and so forth.



HOW CEPH SCALES

In traditional architectures, clients talk to a centralized component (e.g., a gateway, broker, API, facade, etc.), which acts as a single point of entry to a complex subsystem. This imposes a limit to both performance and scalability, while introducing a single point of failure (i.e., if the centralized component goes down, the whole system goes down, too). Ceph eliminates this problem.

CRUSH BACKGROUND

Key to Ceph's design is the autonomous, self-healing, and intelligent Object Storage Daemon (OSD). Storage clients and OSDs both use the CRUSH algorithm to efficiently compute information about data containers on demand, instead of having to depend on a central lookup table. CRUSH provides a better data management mechanism compared to older approaches, and enables massive scale by cleanly distributing the work to all the clients and OSDs in the cluster. CRUSH uses intelligent data replication to ensure resiliency, which is better suited to hyper-scale storage. Let's take a deeper look at how CRUSH works to enable modern cloud storage infrastructures.

CLUSTER MAP

Ceph depends upon clients and OSDs having knowledge of the cluster topology, which is inclusive of 5 maps collectively referred to as the "Cluster Map":

1. **The Monitor Map:** Contains the cluster `fsid`, the position, name address and port of each monitor. It also indicates the current epoch, when the map was created, and the last time it changed. To view a monitor map, execute `ceph mon dump`.
2. **The OSD Map:** Contains the cluster `fsid`, when the map was created and last modified, a list of pools, replica sizes, PG numbers, a list of OSDs and their status (e.g., `up`, `in`). To view an OSD map, execute `ceph osd dump`.
3. **The PG Map:** Contains the PG version, its time stamp, the last OSD map epoch, the full ratios, and details on each placement group such as the PG ID, the *Up Set*, the *Acting Set*, the state of the PG (e.g., `active + clean`), and data usage statistics for each pool.
4. **The CRUSH Map:** Contains a list of storage devices, the failure domain hierarchy (e.g., device, host, rack, row, room, etc.), and rules for traversing the hierarchy when storing data. To view a CRUSH map, execute `ceph osd getcrushmap -o {filename}`; then, decompile it by executing `crushtool -d {comp-crushmap-filename} -o {decomp-crushmap-filename}`. You can view the decompiled map in a text editor or with `cat`.
5. **The MDS Map:** Contains the current MDS map epoch, when the map was created, and the last time it changed. It also contains the pool for storing metadata, a list of metadata servers, and which metadata servers are `up` and `in`. To view an MDS map, execute `ceph mds dump`.

Each map maintains an iterative history of its operating state changes, which enables Ceph to monitor the cluster. The maps that are the most relevant to scalability include the CRUSH Map, the OSD Map, and the PG Map.

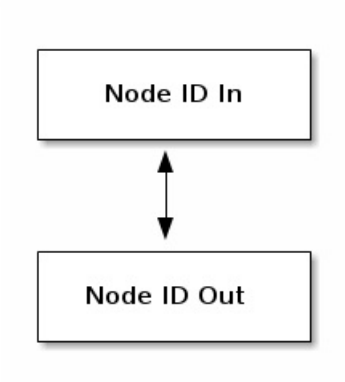
MONITOR QUORUMS

Ceph's monitors maintain a master copy of the cluster map. So Ceph daemons and clients merely contact the monitor periodically to ensure they have the most recent copy of the cluster map. Ceph monitors are light-weight processes, but for added reliability and fault tolerance, Ceph supports distributed monitors. Ceph must have agreement among various monitor instances regarding the state of the cluster. To establish a consensus, Ceph always uses a majority of monitors (e.g., 1, 3-*n*, etc.) and the Paxos algorithm in order to establish a consensus.

For details on configuring monitors, see the Monitor Config Reference.

SMART DAEMONS

Ceph's cluster map determines whether a node in a network is `in` the Ceph cluster or `out` of the Ceph cluster.



In many clustered architectures, the primary purpose of cluster membership is so that a centralized interface knows which hosts it can access. Ceph takes it a step further: Ceph's nodes are cluster aware. Each node knows about other nodes in the cluster. This enables Ceph's monitor, OSD, and metadata server daemons to interact directly with each other. One major benefit of this approach is that Ceph can utilize the CPU and RAM of its nodes to easily perform tasks that would bog down a centralized server.

Ceph OSDs join a cluster and report on their status. At the lowest level, the OSD status is `up` or `down` reflecting whether or not it

is running and able to service requests. If an OSD is `down` and `in` the cluster, this status may indicate the failure of the OSD.

With peer awareness, OSDs can communicate with other OSDs and monitors to perform tasks. OSDs take client requests to read data from or write data to pools, which have placement groups. When a client makes a request to write data to a primary OSD, the primary OSD knows how to determine which OSDs have the placement groups for the replica copies, and then update those OSDs. This means that OSDs can also take requests from other OSDs. With multiple replicas of data across OSDs, OSDs can also "peer" to ensure that the placement groups are in sync. See Monitoring OSDs and PGs for additional details.

If an OSD is not running (e.g., it crashes), the OSD cannot notify the monitor that it is `down`. The monitor can ping an OSD periodically to ensure that it is running. However, Ceph also empowers OSDs to determine if a neighboring OSD is `down`, to update the cluster map and to report it to the monitor(s). When an OSD is `down`, the data in the placement group is said to be `degraded`. If the OSD is `down` and `in`, but subsequently taken `out` of the cluster, the OSDs receive an update to the cluster map and rebalance the placement groups within the cluster automatically. See Heartbeats for additional details.

CALCULATING PG IDS

When a Ceph client binds to a monitor, it retrieves the latest copy of the cluster map. With the cluster map, the client knows about all of the monitors, OSDs, and metadata servers in the cluster. However, it doesn't know anything about object locations. Object locations get computed.

The only input required by the client is the object ID and the pool. It's simple: Ceph stores data in named pools (e.g., "liverpool"). When a client wants to store a named object (e.g., "john," "paul," "george," "ringo", etc.) it calculates a placement group using the object name, a hash code, the number of OSDs in the cluster and the pool name. Ceph clients use the following steps to compute PG IDs.

1. The client inputs the pool ID and the object ID. (e.g., pool = "liverpool" and object-id = "john")
2. CRUSH takes the object ID and hashes it.
3. CRUSH calculates the hash modulo the number of OSDs. (e.g., `0x58`) to get a PG ID.
4. CRUSH gets the pool ID given the pool name (e.g., "liverpool" = 4)
5. CRUSH prepends the pool ID to the pool ID to the PG ID (e.g., `4.0x58`).

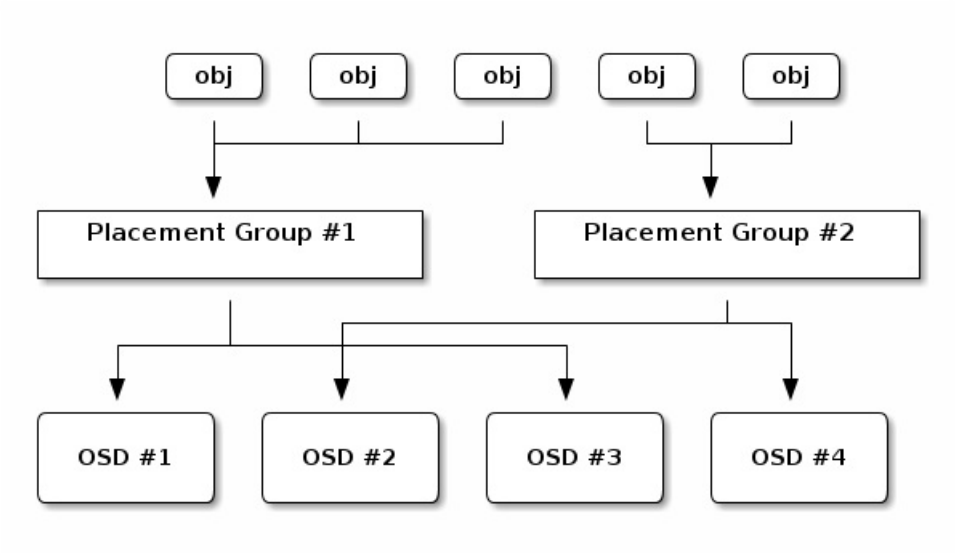
Computing object locations is much faster than performing object location query over a chatty session. The CRUSH algorithm allows a client to compute where objects *should* be stored, and enables the client to contact the primary OSD to store or retrieve the objects.

ABOUT POOLS

The Ceph storage system supports the notion of 'Pools', which are logical partitions for storing objects. Pools set ownership/access, the number of object replicas, the number of placement groups, and the CRUSH rule set to use. Each pool has a number of placement groups that are mapped dynamically to OSDs. When clients store objects, CRUSH maps each object to a placement group.

MAPPING PGS TO OSDS

Mapping objects to placement groups instead of directly to OSDs creates a layer of indirection between the OSD and the client. The cluster must be able to grow (or shrink) and rebalance where it stores objects dynamically. If the client "knew" which OSD had which object, that would create a tight coupling between the client and the OSD. Instead, the CRUSH algorithm maps each object to a placement group and then maps each placement group to one or more OSDs. This layer of indirection allows Ceph to rebalance dynamically when new OSDs come online. The following diagram depicts how CRUSH maps objects to placement groups, and placement groups to OSDs.

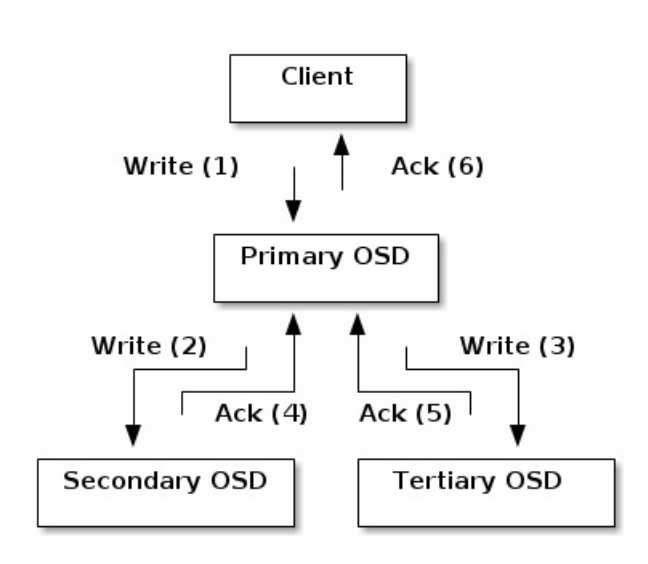


With a copy of the cluster map and the CRUSH algorithm, the client can compute exactly which OSD to use when reading or writing a particular object.

CLUSTER-SIDE REPLICATION

The OSD daemon also uses the CRUSH algorithm, but the OSD daemon uses it to compute where replicas of objects should be stored (and for rebalancing). In a typical write scenario, a client uses the CRUSH algorithm to compute where to store an object, maps the object to a pool and placement group, then looks at the CRUSH map to identify the primary OSD for the placement group.

The client writes the object to the identified placement group in the primary OSD. Then, the primary OSD with its own copy of the CRUSH map identifies the secondary and tertiary OSDs for replication purposes, and replicates the object to the appropriate placement groups in the secondary and tertiary OSDs (as many OSDs as additional replicas), and responds to the client once it has confirmed the object was stored successfully.



Since any network device has a limit to the number of concurrent connections it can support, a centralized system has a low physical limit at high scales. By enabling clients to contact nodes directly, Ceph increases both performance and total system capacity simultaneously, while removing a single point of failure. Ceph clients can maintain a session when they need to, and with a particular OSD instead of a centralized server. For a detailed discussion of CRUSH, see CRUSH - Controlled, Scalable, Decentralized Placement of Replicated Data.

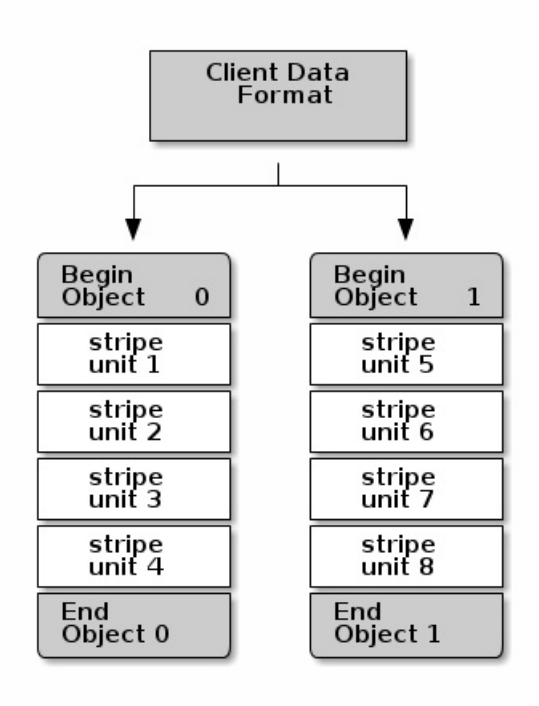
EXTENDING CEPH

Todo: explain “classes”

HOW CEPH CLIENTS STRIPE DATA

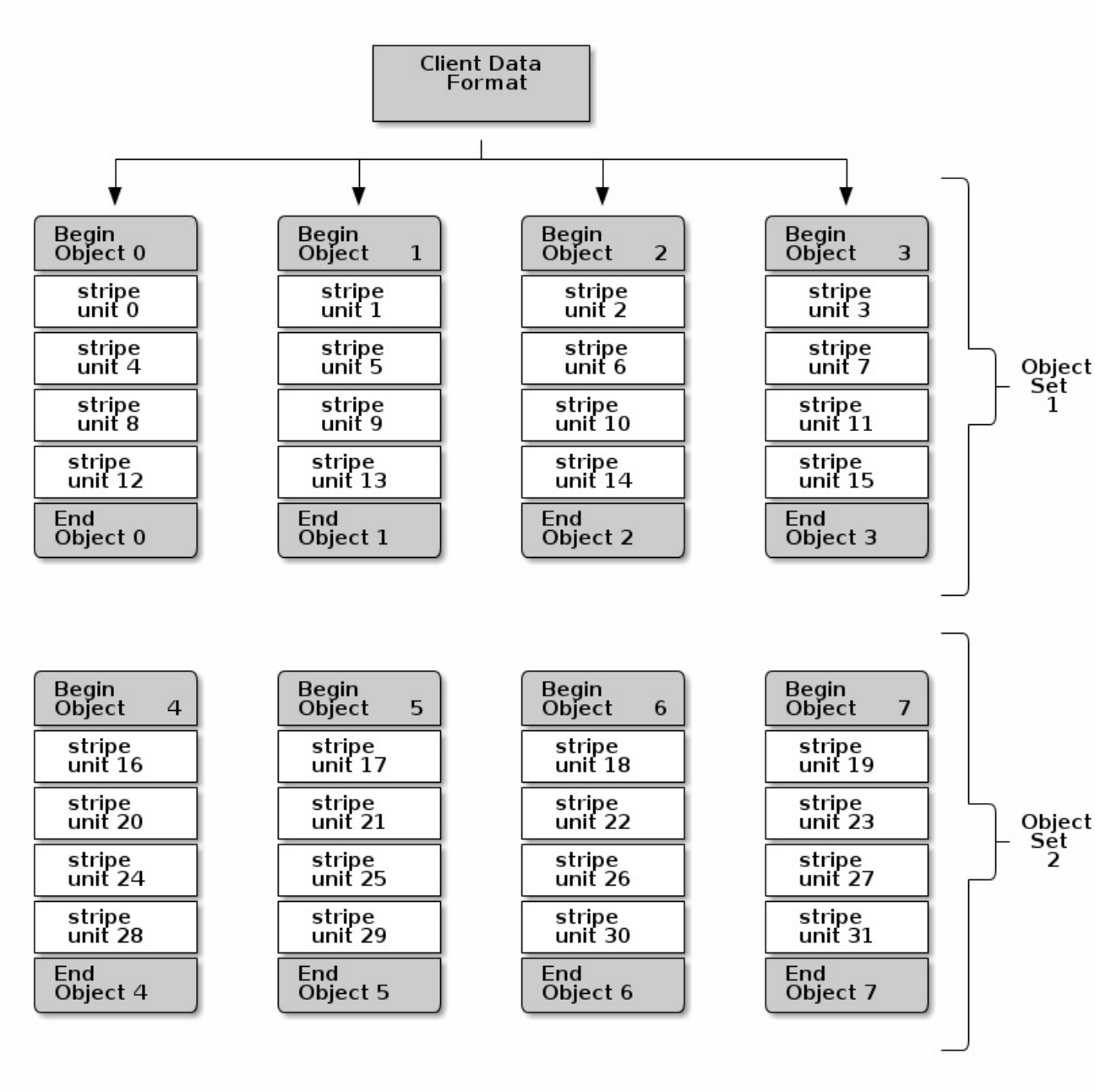
Storage devices have throughput limitations, which impact performance and scalability. So storage systems often support striping–storing sequential pieces of information across across multiple storage devices–to increase throughput and performance. The most common form of data striping comes from RAID. The RAID type most similar to Ceph's striping is RAID 0, or a 'striped volume.' Ceph's striping offers the throughput of RAID 0 striping, the reliability of n-way RAID mirroring and faster recovery.

Ceph provides three types of clients: block device, CephFS filesystem, and Gateway. A Ceph client converts its data from the representation format it provides to its users (a block device image, RESTful objects, CephFS filesystem directories) into objects for storage in the Object Store. The simplest Ceph striping format involves a stripe count of 1 object. Clients write stripe units to an object until the object is at its maximum capacity, and then create another object for additional stripes of data. The simplest form of striping may be sufficient for small block device images, S3 or Swift objects, or CephFS files. However, this simple form doesn't take maximum advantage of Ceph's ability to distribute data across placement groups, and consequently doesn't improve performance very much. The following diagram depicts the simplest form of striping:



If you anticipate large images sizes, large S3 or Swift objects (video), or large CephFS directories, you may see considerable read/write performance improvements by striping client data over mulitple objects within an object set. Significant write performance occurs when the client writes the stripe units to their corresponding objects in parallel. Since objects get mapped to different placement groups and further mapped to different OSDs, each write occurs in parallel at the maximum write speed. A write to a single disk would be limited by the head movement (e.g. 6ms per seek) and bandwidth of that one device (e.g. 100MB/s). By spreading that write over multiple objects (which map to different placement groups and OSDs) Ceph can reduce the number of seeks per drive and combine the throughput of multiple drives to achieve much faster write (or read) speeds.

In the following diagram, client data gets striped across an object set (`object set 1` in the following diagram) consisting of 4 objects, where the first stripe unit is `stripe unit 0` in `object 0`, and the fourth stripe unit is `stripe unit 3` in `object 3`. After writing the fourth stripe, the client determines if the object set is full. If the object set is not full, the client begins writing a stripe to the first object again (`object 0` in the following diagram). If the object set is full, the client creates a new object set (`object set 2` in the following diagram), and begins writing to the first stripe (`stripe unit 16`) in the first object in the new object set (`object 4` in the diagram below).



Three important variables determine how Ceph stripes data:

- **Object Size:** Objects in the Ceph Object Store have a maximum configurable size (e.g., 2MB, 4MB, etc.). The object size should be large enough to accomodate many stripe units, and should be a multiple of the stripe unit.
- **Stripe Width:** Stripes have a configurable unit size (e.g., 64kb). The Ceph client divides the data it will write to objects into equally sized stripe units, except for the last stripe unit. A stripe width, should be a fraction of the Object Size so that an object may contain many stripe units.
- **Stripe Count:** The Ceph client writes a sequence of stripe units over a series of objects determined by the stripe count. The series of objects is called an object set. After the Ceph client writes to the last object in the object set, it returns to the first object in the object set.

Important: Test the performance of your striping configuration before putting your cluster into production. You CANNOT change these striping parameters after you stripe the data and write it to objects.

Once the Ceph client has striped data to stripe units and mapped the stripe units to objects, Ceph's CRUSH algorithm maps the objects to placement groups, and the placement groups to OSDs before the objects are stored as files on a storage disk. See How Ceph Scales for details.

Important: Striping is independent of object replicas. Since CRUSH replicates objects across OSDs, stripes get replicated automatically.

S3/Swift Objects and Object Store Objects Compared

Ceph's Gateway uses the term *object* to describe the data it stores. S3 and Swift objects from the Gateway are not the same as the objects Ceph writes to the Object Store. Gateway objects are mapped to Ceph objects that get written to the Object Store. The S3 and Swift objects do not necessarily correspond in a 1:1 manner with an object stored in the Object Store. It is

possible for an S3 or Swift object to map to multiple Ceph objects.

Note: Since a client writes to a single pool, all data striped into objects get mapped to placement groups in the same pool. So they use the same CRUSH map and the same access controls.

Tip: The objects Ceph stores in the Object Store are not striped. RGW, RBD and CephFS automatically stripe their data over multiple RADOS objects. Clients that write directly to the Object Store via `librados` must peform the the striping (and parallel I/O) for themselves to obtain these benefits.

DATA CONSISTENCY

As part of maintaining data consistency and cleanliness, Ceph OSDs can also scrub objects within placement groups. That is Ceph OSDs can compare object metadata in one placement group with its replicas in placement groups stored in other OSDs. Scrubbing (usually performed daily) catches OSD bugs or filesystem errors. OSDs can also perform deeper scrubbing by comparing data in objects bit-for-bit. Deep scrubbing (usually performed weekly) finds bad sectors on a disk that weren't apparent in a light scrub.

See Data Scrubbing for details on configuring scrubbing.

METADATA SERVERS

The Ceph filesystem service is provided by a daemon called `ceph-mds`. It uses RADOS to store all the filesystem metadata (directories, file ownership, access modes, etc), and directs clients to access RADOS directly for the file contents. The Ceph filesystem aims for POSIX compatibility. `ceph-mds` can run as a single process, or it can be distributed out to multiple physical machines, either for high availability or for scalability.

- **High Availability**: The extra `ceph-mds` instances can be *standby*, ready to take over the duties of any failed `ceph-mds` that was *active*. This is easy because all the data, including the journal, is stored on RADOS. The transition is triggered automatically by `ceph-mon`.
- **Scalability**: Multiple `ceph-mds` instances can be *active*, and they will split the directory tree into subtrees (and shards of a single busy directory), effectively balancing the load amongst all *active* servers.

Combinations of *standby* and *active* etc are possible, for example running 3 *active* `ceph-mds` instances for scaling, and one *standby* intance for high availability.

CLIENT INTERFACES

AUTHENTICATION AND AUTHORIZATION

Ceph clients can authenticate their users with Ceph monitors, OSDs and metadata servers. Authenticated users gain authorization to read, write and execute Ceph commands. The Cephx authentication system is similar to Kerberos, but avoids a single point of failure to ensure scalability and high availability. For details on Cephx, see Ceph Authentication and Authorization.

LIBRADOS

Todo: Snapshotting, Import/Export, Backup

Todo: native APIs

RBD

RBD stripes a block device image over multiple objects in the cluster, where each object gets mapped to a placement group and distributed, and the placement groups are spread across separate `ceph-osd` daemons throughout the cluster.

Important: Striping allows RBD block devices to perform better than a single server could!

RBD's thin-provisioned snapshottable block devices are an attractive option for virtualization and cloud computing. In virtual

machine scenarios, people typically deploy RBD with the `rbd` network storage driver in Qemu/KVM, where the host machine uses `librbd` to provide a block device service to the guest. Many cloud computing stacks use `libvirt` to integrate with hypervisors. You can use RBD thin-provisioned block devices with Qemu and libvirt to support OpenStack and CloudStack among other solutions.

While we do not provide `librbd` support with other hypervisors at this time, you may also use RBD kernel objects to provide a block device to a client. Other virtualization technologies such as Xen can access the RBD kernel object(s). This is done with the command-line tool `rbd`.

RGW

The RADOS Gateway daemon, `radosgw`, is a FastCGI service that provides a RESTful HTTP API to store objects and metadata. It layers on top of RADOS with its own data formats, and maintains its own user database, authentication, and access control. The RADOS Gateway uses a unified namespace, which means you can use either the OpenStack Swift-compatible API or the Amazon S3-compatible API. For example, you can write data using the S3-comptable API with one application and then read data using the Swift-compatible API with another application.

See RADOS Gateway for details.

CEPHFS

Todo: cephfs, ceph-fuse

LIMITATIONS OF PRIOR ART

Today's storage systems have demonstrated an ability to scale out, but with some significant limitations: interfaces, session managers, and stateful sessions with a centralized point of access often limit the scalability of today's storage architectures. Furthermore, a centralized interface that dispatches requests from clients to server nodes within a cluster and subsequently routes responses from those server nodes back to clients will hit a scalability and/or performance limitation.

Another problem for storage systems is the need to manually rebalance data when increasing or decreasing the size of a data cluster. Manual rebalancing works fine on small scales, but it is a nightmare at larger scales because hardware additions are common and hardware failure becomes an expectation rather than an exception when operating at the petabyte scale and beyond.

The operational challenges of managing legacy technologies with the burgeoning growth in the demand for unstructured storage makes legacy technologies inadequate for scaling into petabytes. Some legacy technologies (e.g., SAN) can be considerably more expensive, and more challenging to maintain when compared to using commodity hardware. Ceph uses commodity hardware, because it is substantially less expensive to purchase (or to replace), and it only requires standard system administration skills to use it.