# Ceph Mon: a technical overview

Paolo VIOTTI

EURECOM

27th March 2015

**Abstract**

Ceph is a free software storage platform designed to provide object, block and file storage using computer clusters running on commodity hardware. Ceph's main design goals include high scalability, fault tolerance and low maintenance requirements. This document provides an in-depth technical overview of the design of Ceph Monitor, i.e. the Ceph component in charge of maintaining a map of the cluster along with authorization information.

## 1 Ceph: an introduction

Ceph [1] is a free software storage platform that provides object, block and file storage using computer clusters running on commodity hardware. Since its origin as a research project around 2006, Ceph has undergone constant and substantial development, thus gaining popularity which is reflected by an ever increasing adoption as storage backend in state-of-the-art high-end computing systems [2].

As illustrated in Fig. 1, Ceph exposes to application clients multiple APIs:

- a POSIX-compatible distributed file system (**Ceph FS**), built as Linux kernel module or user-space FUSE client;
- a REST-based storage gateway (**RADOSGW**) compatible with OpenStack Swift and Amazon S3;
- a block storage device (**RDB**) suitable for virtualization platforms making use of kernel virtualization technologies such as QEMU or KVM.

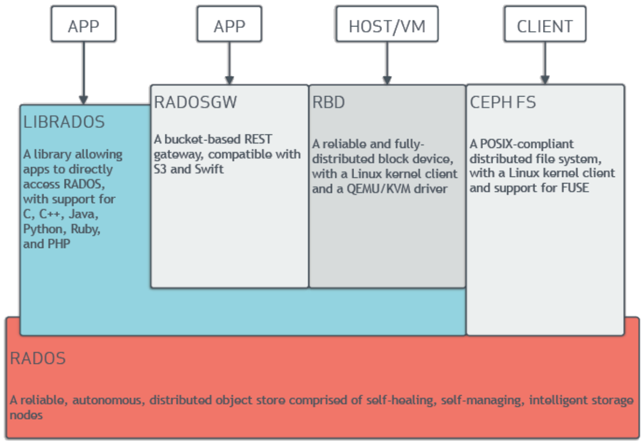
APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
RADOS
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

Figure 1: Ceph application stack (courtesy of ceph.com)

Beside, an application developer may even hook into the low level API exposed through **librados** and offered in several programming languages in order to directly connect with **RADOS** (*Reliable Autonomic Distributed Object Store*), i.e. the inner object storage layer that acts as foundation of the interfaces mentioned above.

The Ceph cluster consists of different components, running as distributed daemons:

- Cluster monitors (*ceph-mon*, or Mon) that keep track of active and failed cluster nodes;
- Metadata servers (*ceph-mds*) that store the metadata of inodes and directories;
- Object storage devices (*ceph-osd*) that actually store data on local filesystems;
- RESTful gateways (*ceph-rgw*) that expose the object storage layer as an interface compatible with Amazon S3 or OpenStack Swift APIs.

The rest of this document focuses on the monitor component, starting from its high-level architecture, deep down to the implementation details present, to date, in its C++ codebase [3]. To marry the need for this 1000 ft. view with the requirement of presenting the related low level features, in the following sections informal functional descriptions may be presented beside the name of their main corresponding entities in the code (e.g. `classes` or *data structures*). Most of the code required to run the monitor is contained in the `src/mon` directory of the repository [3] (∼34k LOC), although it shares with the rest of the codebase few functions and classes related to specific parts of the system (e.g. OSD or MDS daemons, respectively in `src/osd` and `src/mds`), or networking, logging and other basic facilities.

# 2 Ceph Mon

A Ceph Monitor maintains a set of structured information about the cluster state, including:

- the monitor map - `MonMap`;
- the OSD map - `OSDMap`;
- the Placement Group (PG) map - `PGMap`;
- the MDS map - `MDSMap`.

Additionally, information about authorization and capabilities granted to users over the different components of the system is maintained. Besides, Ceph holds a history, as a sequence of *epochs*, of each state change in the maps. All these information are replicated across the ceph-mon instances - which are, for example, 3 or 5, deployed on different machines. Each monitor replica keeps its data strongly consistent using an implementation of the Paxos consensus algorithm (`Paxos`).

## 2.1 Architecture

As illustrated in Fig. 2, the Ceph monitor is composed by several sub-monitors which oversee different aspects of the cluster. All these monitors refer to a unique instance of Paxos[1] in order to establish a total order between possibly concurrent operations and achieve strong consistency. The Paxos instance is agnostic of any semantic associated with the updates it helps ordering, as in fact it just delivers blobs (i.e. *bufferlist* as payload of `Message`).

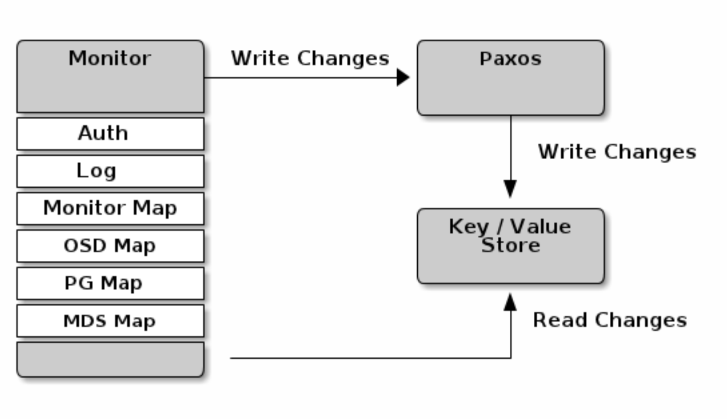

Figure 2: Ceph Monitor high-level architecture (courtesy of ceph.com)

[1]This was not true before v0.58, see: `http://ceph.com/dev-notes/cephs-new-monitor-changes/`

## 2.2 Implementation details

As depicted in Figure 3, the `Monitor` object acts as container and coordinator of different objects.

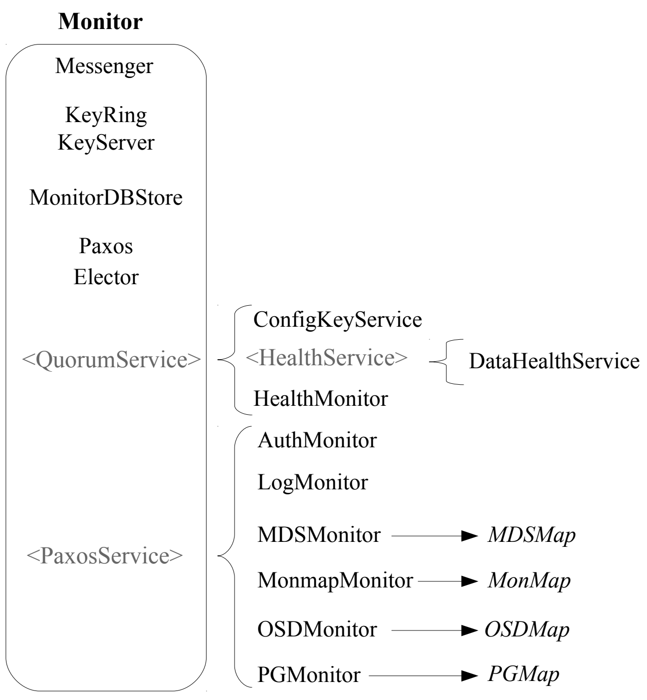

Figure 3: Ceph Monitor classes

Some of them inherit from a common parent class named `QuorumService`. In particular, `HealthMonitor` holds a reference to `DataHealthService`, which is in turn child class of `HealthService`. `DataHealthService` is responsible of periodically sharing with other quorum members or presenting to the admin user some statistics about general cluster health. `KeyConfigService` is another subclass of `QuorumService` which uses `Paxos` in order to propagate to the monitor cluster members the addition or removal of cryptographic material to the local store.

Beside the above mentioned classes, other singleton objects take part in the general functioning of the monitor as explained in the following list.

| | |
|---|---|
| `Messenger` | in charge of the network communications with other parties; |
| `KeyRing`, `KeyServer` | manage keying material for authorization; |
| `MonitorDBStore` | manages the local database (i.e. LevelDB) which stores maps and cryptographic material; |

`Paxos` Paxos implementation;

`Elector` used only to maintain the local state during elections.

Finally, few classes which inherit from `PaxosService` are properly named `*Mon itor` as they implement the logic to maintain the maps of the different sections of the Ceph cluster. Aside from `AuthMonitor`, which is responsible for enabling or disabling capabilities on a user basis, and `LogMonitor`, which tunes logging as required by the administrative user, all other `*Monitor` s refer to specific classes named `*Map` to save their pertaining states. Such maps are also serialized to the local key-value store (i.e. LevelDB) which is managed by `MonitorDBStore`. To guarantee total ordering of updates, every change on maps is submitted to the Paxos service and then applied only once the related Paxos proposal has been committed.

The monitor can assume the following states during an execution:

- *probing*: initial and bootstrap phase state;
- *synchronizing*: used when synchronizing the state due to significant drifting from leader's replica;
- *electing*: assumed when starting or joining a Paxos election;
- *leader*: the monitor is active and leader in the Paxos quorum;
- *peon*: the monitor is active and follower in the Paxos quorum;
- *shutdown*: when shutdown process is ongoing.

## 2.3 Paxos

The `Paxos` class implements the Paxos algorithm in order to guarantee agreement over the ordering of changes performed on Ceph maps. Apart from a couple of online informal or incomplete documents [4, 5], the only reliable source of documentation available about this Paxos implementation is contained in the comments of the Paxos class and header files. In particular, there is included the following excerpt, which reports about the difference between the Paxos algorithms and this practical implementation.

```
This libary is based on the Paxos algorithm, but varies in a few key ways:
 1- Only a single new value is generated at a time,
    simplifying the recovery logic.
 2- Nodes track "committed" values,
     and share them generously (and trustingly)
 3- A 'leasing' mechanism is built-in, allowing nodes to determine
    when it is safe to "read" their copy of the last committed value.
```

Paxos tracks the last and the first committed key number on the store, along with each numbered key and their corresponding values that have been committed thus far. As mentioned before, values are binary blocks opaque to

Paxos. A trimming mechanism guarantees that the number of Paxos states kept in the local databases is limited according to a configuration paramenter.

The state diagram shown in Figure 4 presents the states Ceph's Paxos can assume and their transitions. We note that not every state is achievable by peons - states marked in blue are assumed only by the leader. Several callback functions are used by `PaxosService` s to be notified of `Paxos` state changes.

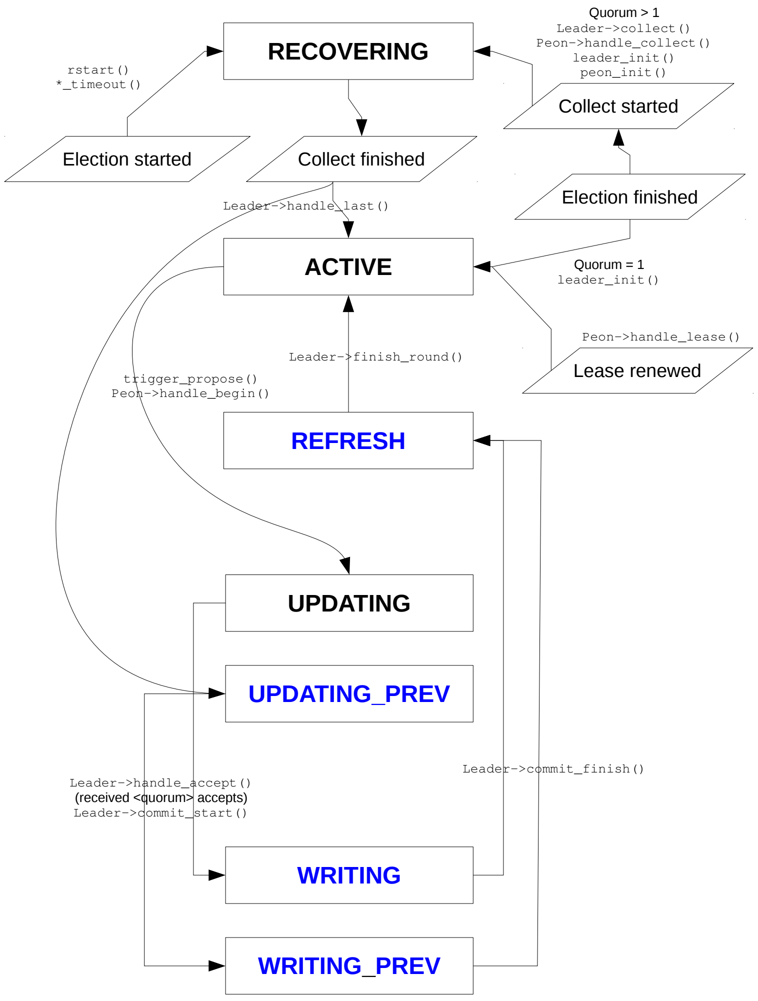

Figure 4: Ceph's Paxos state diagram.
Blue states can be achieved only by the leader.

In Figure 5 we show the sequence diagram of an ideal Ceph's Paxos execution. The `do_refresh()` function is supposed to let local `PaxosService` s know of the new operations just committed.

The so-called *collect* phase can be mapped onto the original Paxos' *prepare* phase. In this phase the leader will generate a proposal number, taking the old proposal numbers into consideration, and it will send it to a quorum, along with its first and last committed versions. By sending these information in a message to the quorum, it expects to obtain acceptances from a majority, allowing it to proceed, or be informed of a higher proposal number known by one or more of the Peons in the quorum. In fact, if a peon replies with a higher proposal number, the leader will have to abort the current proposal in order to retry with the proposal number specified by the peon. The leader will also check if peons replied with accepted but yet uncommitted values. In that case, if its version is higher than leader's last committed value by one, the leader assumes that the peons knows a value from a previous proposal that has never been committed, so it tries to commit that value by proposing it next.

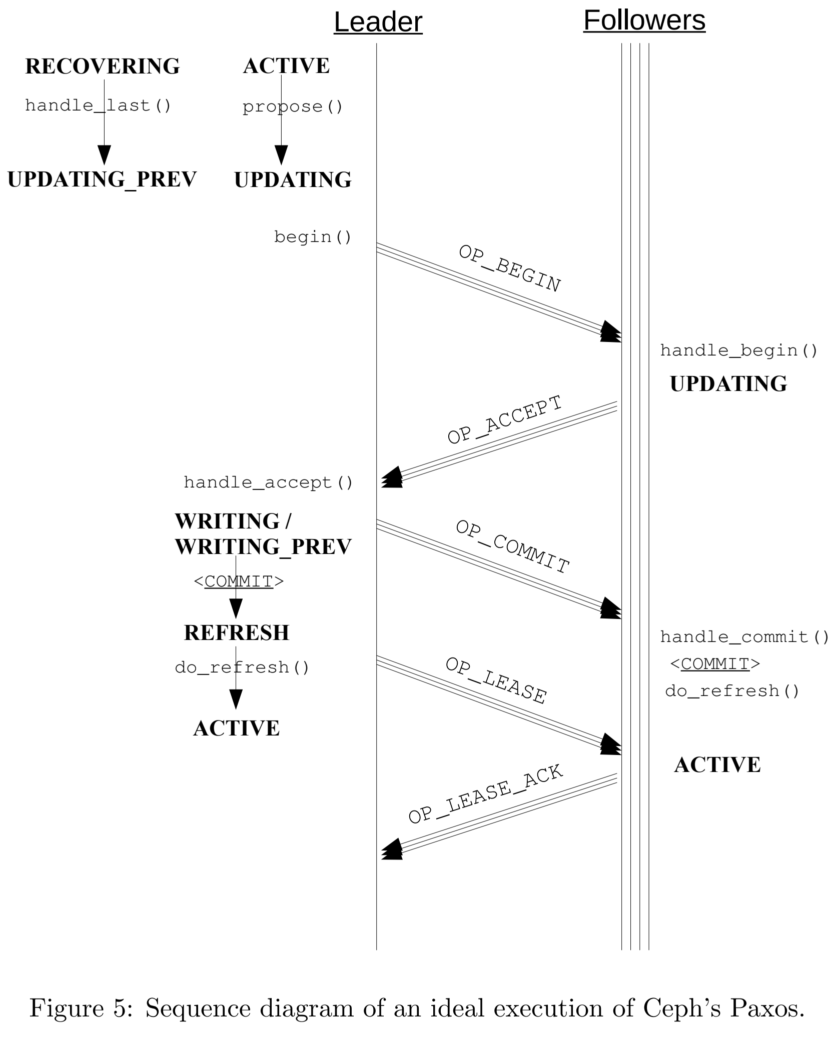

Figure 5: Sequence diagram of an ideal execution of Ceph's Paxos.

In general, when a timeout is triggered at any step of the algorithm, `Paxos` cancels pending proposals and goes back to `Monitor` which starts a *probe* phase to reach out to other monitors. As soon as the `Monitor` collects enough probe replies, it spawn a Paxos election using `Elector`.

The Paxos instance can be *readble* or *writable* by its local services. Paxos versions are not readable when electing, when it has not committed values or the lease is currently not valid. Paxos is writable when it is leader, it has a valid lease and it is in the *active* state.

Finally, the *leases* implemented in Ceph's Paxos are intended to ensure that clients and `PaxosService` s are bound to read fresh information from local replicas. As reported in the official Ceph documentation [6], it is important to remark that leases and, more generally, <u>the correctness of the whole Paxos implementation depends on the synchronization of clocks</u> of the machines hosting the monitor replicas. Thus, using tools such as NTP is necessary to *minimize* the chances of incorrect behaviour.

## 2.4 Additional notes

```
To ask / understand:

 - MonitorStore.{h,cc} never used: obsolete?
 - difference between MonCommands and DumplingMonCommands?
 - functional difference between QuorumService and PaxosService
    (since they both contain refererences to Paxos and the Monitor)?
 - why does Paxos need the lease to be writable?
 - Paxos.is_writeable is never used !
 - in Paxos.h COMMENTS ON commit_start(), commit_finish(),
    handle_accept() and handle_last() ABOUT STATE CHANGES
    ARE WRONG / OBSOLETE!!
    In fact it changes from UPDATE{PREV} to WRITING{PREV},
    not ACTIVE straightaway
```

## References

[1] Ceph storage platform. `http://ceph.com/`

[2] OpenStack User Survey Insights: November 2014. `http://perma.cc/367D-G5YN`

[3] Ceph source code. `https://github.com/ceph/ceph`

[4] Ceph's new monitor changes. `http://ceph.com/dev-notes/cephs-new-monitor-changes/` - March 7th, 2013

[5] Monitors and Paxos, a chat with Joao. `http://ceph.com/community/monitors-and-paxos-a-chat-with-joao/` - September 10th, 2013

[6] Monitor issues and wall clock skews. `http://ceph.com/docs/master/rados/troubleshooting/troubleshooting-mon/#clock-skews`