TROUBLESHOOTING OSDS

Before troubleshooting your OSDs, check your monitors and network first. If you execute `ceph health` or `ceph -s` on the command line and Ceph returns a health status, it means that the monitors have a quorum. If you don't have a monitor quorum or if there are errors with the monitor status, address the monitor issues first. Check your networks to ensure they are running properly, because networks may have a significant impact on OSD operation and performance.

OBTAINING DATA ABOUT OSDS

A good first step in troubleshooting your OSDs is to obtain information in addition to the information you collected while monitoring your OSDs (e.g., `ceph osd tree`).

CEPH LOGS

If you haven't changed the default path, you can find Ceph log files at `/var/log/ceph`:

```
ls /var/log/ceph
```

If you don't get enough log detail, you can change your logging level. See Logging and Debugging for details to ensure that Ceph performs adequately under high logging volume.

ADMIN SOCKET

Use the admin socket tool to retrieve runtime information. For details, list the sockets for your Ceph processes:

```
ls /var/run/ceph
```

Then, execute the following, replacing `{daemon-name}` with an actual daemon (e.g., `osd.0`):

```
ceph daemon osd.0 help
```

Alternatively, you can specify a `{socket-file}` (e.g., something in `/var/run/ceph`):

```
ceph daemon {socket-file} help
```

The admin socket, among other things, allows you to:

- List your configuration at runtime
- Dump historic operations
- Dump the operation priority queue state
- Dump operations in flight
- Dump perfcounters

DISPLAY FREESPACE

Filesystem issues may arise. To display your filesystem's free space, execute `df`.

```
df -h
```

Execute `df --help` for additional usage.

I/O STATISTICS

Use iostat to identify I/O-related issues.

```
iostat -x
```

DIAGNOSTIC MESSAGES

To retrieve diagnostic messages, use `dmesg` with `less`, `more`, `grep` or `tail`. For example:

```
dmesg | grep scsi
```

STOPPING W/OUT REBALANCING

Periodically, you may need to perform maintenance on a subset of your cluster, or resolve a problem that affects a failure domain (e.g., a rack). If you do not want CRUSH to automatically rebalance the cluster as you stop OSDs for maintenance, set the cluster to `noout` first:

```
ceph osd set noout
```

Once the cluster is set to `noout`, you can begin stopping the OSDs within the failure domain that requires maintenance work.

```
stop ceph-osd id={num}
```

Note: Placement groups within the OSDs you stop will become degraded while you are addressing issues with within the failure domain.

Once you have completed your maintenance, restart the OSDs.

```
start ceph-osd id={num}
```

Finally, you must unset the cluster from `noout`.

```
ceph osd unset noout
```

OSD NOT RUNNING

Under normal circumstances, simply restarting the `ceph-osd` daemon will allow it to rejoin the cluster and recover.

AN OSD WON'T START

If you start your cluster and an OSD won't start, check the following:

- **Configuration File:** If you were not able to get OSDs running from a new installation, check your configuration file to ensure it conforms (e.g., `host` not `hostname`, etc.).
- **Check Paths:** Check the paths in your configuration, and the actual paths themselves for data and journals. If you separate the OSD data from the journal data and there are errors in your configuration file or in the actual mounts, you may have trouble starting OSDs. If you want to store the journal on a block device, you should partition your journal disk and assign one partition per OSD.
- **Check Max Threadcount:** If you have a node with a lot of OSDs, you may be hitting the default maximum number of threads (e.g., usually 32k), especially during recovery. You can increase the number of threads using `sysctl` to see if increasing the maximum number of threads to the maximum possible number of threads allowed (i.e., 4194303) will help. For example:

  ```
  sysctl -w kernel.pid_max=4194303
  ```

If increasing the maximum thread count resolves the issue, you can make it permanent by including a `kernel.pid_max` setting in the `/etc/sysctl.conf` file. For example:

```
kernel.pid_max = 4194303
```

- **Kernel Version:** Identify the kernel version and distribution you are using. Ceph uses some third party tools by default, which may be buggy or may conflict with certain distributions and/or kernel versions (e.g., Google perftools). Check the OS recommendations to ensure you have addressed any issues related to your kernel.
- **Segment Fault:** If there is a segment fault, turn your logging up (if it is not already), and try again. If it segment faults again, contact the ceph-devel email list and provide your Ceph configuration file, your monitor output and the contents of your log file(s).

AN OSD FAILED

When a `ceph-osd` process dies, the monitor will learn about the failure from surviving `ceph-osd` daemons and report it via the `ceph health` command:

```
ceph health
HEALTH_WARN 1/3 in osds are down
```

Specifically, you will get a warning whenever there are `ceph-osd` processes that are marked `in` and `down`. You can identify which `ceph-osds` are `down` with:

```
ceph health detail
HEALTH_WARN 1/3 in osds are down
osd.0 is down since epoch 23, last address 192.168.106.220:6800/11080
```

If there is a disk failure or other fault preventing `ceph-osd` from functioning or restarting, an error message should be present in its log file in `/var/log/ceph`.

If the daemon stopped because of a heartbeat failure, the underlying kernel file system may be unresponsive. Check `dmesg` output for disk or other kernel errors.

If the problem is a software error (failed assertion or other unexpected error), it should be reported to the ceph-devel email list.

NO FREE DRIVE SPACE

Ceph prevents you from writing to a full OSD so that you don't lose data. In an operational cluster, you should receive a warning when your cluster is getting near its full ratio. The `mon osd full ratio` defaults to `0.95`, or 95% of capacity before it stops clients from writing data. The `mon osd backfillfull ratio` defaults to `0.90`, or 90 % of capacity when it blocks backfills from starting. The `mon osd nearfull ratio` defaults to `0.85`, or 85% of capacity when it generates a health warning.

Full cluster issues usually arise when testing how Ceph handles an OSD failure on a small cluster. When one node has a high percentage of the cluster's data, the cluster can easily eclipse its nearfull and full ratio immediately. If you are testing how Ceph reacts to OSD failures on a small cluster, you should leave ample free disk space and consider temporarily lowering the `mon osd full ratio`, `mon osd backfillfull ratio` and `mon osd nearfull ratio`.

Full `ceph-osds` will be reported by `ceph health`:

```
ceph health
HEALTH_WARN 1 nearfull osd(s)
```

Or:

```
ceph health detail
HEALTH_ERR 1 full osd(s); 1 backfillfull osd(s); 1 nearfull osd(s)
osd.3 is full at 97%
osd.4 is backfill full at 91%
osd.2 is near full at 87%
```

The best way to deal with a full cluster is to add new `ceph-osds`, allowing the cluster to redistribute data to the newly available storage.

If you cannot start an OSD because it is full, you may delete some data by deleting some placement group directories in the full OSD.

Important: If you choose to delete a placement group directory on a full OSD, **DO NOT** delete the same placement group directory on another full OSD, or **YOU MAY LOSE DATA**. You **MUST** maintain at least one copy of your data on at least one OSD.

See Monitor Config Reference for additional details.

OSDS ARE SLOW/UNRESPONSIVE

A commonly recurring issue involves slow or unresponsive OSDs. Ensure that you have eliminated other troubleshooting possibilities before delving into OSD performance issues. For example, ensure that your network(s) is working properly and your OSDs are running. Check to see if OSDs are throttling recovery traffic.

Tip: Newer versions of Ceph provide better recovery handling by preventing recovering OSDs from using up system resources so that up and `in` OSDs are not available or are otherwise slow.

NETWORKING ISSUES

Ceph is a distributed storage system, so it depends upon networks to peer with OSDs, replicate objects, recover from faults and check heartbeats. Networking issues can cause OSD latency and flapping OSDs. See Flapping OSDs for details.

Ensure that Ceph processes and Ceph-dependent processes are connected and/or listening.

```
netstat -a | grep ceph
netstat -l | grep ceph
sudo netstat -p | grep ceph
```

Check network statistics.

```
netstat -s
```

DRIVE CONFIGURATION

A storage drive should only support one OSD. Sequential read and sequential write throughput can bottleneck if other processes share the drive, including journals, operating systems, monitors, other OSDs and non-Ceph processes.

Ceph acknowledges writes *after* journaling, so fast SSDs are an attractive option to accelerate the response time–particularly when using the XFS or `ext4` filesystems. By contrast, the `btrfs` filesystem can write and journal simultaneously. (Note, however, that we recommend against using `btrfs` for production deployments.)

Note: Partitioning a drive does not change its total throughput or sequential read/write limits. Running a journal in a separate partition may help, but you should prefer a separate physical drive.

BAD SECTORS / FRAGMENTED DISK

Check your disks for bad sectors and fragmentation. This can cause total throughput to drop substantially.

CO-RESIDENT MONITORS/OSDS

Monitors are generally light-weight processes, but they do lots of `fsync()`, which can interfere with other workloads, particularly if monitors run on the same drive as your OSDs. Additionally, if you run monitors on the same host as the OSDs, you may incur performance issues related to:

- Running an older kernel (pre-3.0)
- Running Argonaut with an old `glibc`
- Running a kernel with no syncfs(2) syscall.

In these cases, multiple OSDs running on the same host can drag each other down by doing lots of commits. That often leads to the bursty writes.

CO-RESIDENT PROCESSES

Spinning up co-resident processes such as a cloud-based solution, virtual machines and other applications that write data to Ceph while operating on the same hardware as OSDs can introduce significant OSD latency. Generally, we recommend optimizing a host for use with Ceph and using other hosts for other processes. The practice of separating Ceph operations from other applications may help improve performance and may streamline troubleshooting and maintenance.

LOGGING LEVELS

If you turned logging levels up to track an issue and then forgot to turn logging levels back down, the OSD may be putting a lot of logs onto the disk. If you intend to keep logging levels high, you may consider mounting a drive to the default path for logging (i.e., `/var/log/ceph/$cluster-$name.log`).

RECOVERY THROTTLING

Depending upon your configuration, Ceph may reduce recovery rates to maintain performance or it may increase recovery rates to the point that recovery impacts OSD performance. Check to see if the OSD is recovering.

KERNEL VERSION

Check the kernel version you are running. Older kernels may not receive new backports that Ceph depends upon for better performance.

KERNEL ISSUES WITH SYNCFS

Try running one OSD per host to see if performance improves. Old kernels might not have a recent enough version of `glibc` to support `syncfs(2)`.

FILESYSTEM ISSUES

Currently, we recommend deploying clusters with XFS.

We recommend against using btrfs or ext4. The btrfs filesystem has many attractive features, but bugs in the filesystem may lead to performance issues and suprious ENOSPC errors. We do not recommend ext4 because xattr size limitations break our support for long object names (needed for RGW).

For more information, see Filesystem Recommendations.

INSUFFICIENT RAM

We recommend 1GB of RAM per OSD daemon. You may notice that during normal operations, the OSD only uses a fraction of that amount (e.g., 100-200MB). Unused RAM makes it tempting to use the excess RAM for co-resident applications, VMs and so forth. However, when OSDs go into recovery mode, their memory utilization spikes. If there is no RAM available, the OSD performance will slow considerably.

OLD REQUESTS OR SLOW REQUESTS

If a `ceph-osd` daemon is slow to respond to a request, it will generate log messages complaining about requests that are taking too long. The warning threshold defaults to 30 seconds, and is configurable via the `osd op complaint time` option. When this happens, the cluster log will receive messages.

Legacy versions of Ceph complain about ‘old requests`:

```
osd.0 192.168.106.220:6800/18813 312 : [WRN] old request osd_op(client.5099.0:790 fatty_26485
```

New versions of Ceph complain about ‘slow requests`:

```
{date} {osd.num} [WRN] 1 slow requests, 1 included below; oldest blocked for > 30.005692 secs
{date} {osd.num}  [WRN] slow request 30.005692 seconds old, received at {date-time}: osd_op(c
```

Possible causes include:

- A bad drive (check `dmesg` output)
- A bug in the kernel file system bug (check `dmesg` output)
- An overloaded cluster (check system load, iostat, etc.)
- A bug in the `ceph-osd` daemon.

Possible solutions

- Remove VMs Cloud Solutions from Ceph Hosts
- Upgrade Kernel
- Upgrade Ceph
- Restart OSDs

DEBUGGING SLOW REQUESTS

If you run “ceph daemon osd.<id> dump_historic_ops” or “dump_ops_in_flight”, you will see a set of operations and a list of events each operation went through. These are briefly described below.

Events from the Messenger layer:

- header_read: when the messenger first started reading the message off the wire
- throttled: when the messenger tried to acquire memory throttle space to read the message into memory
- all_read: when the messenger finished reading the message off the wire
- dispatched: when the messenger gave the message to the OSD
- Initiated: <This is identical to header_read. The existence of both is a historical oddity.

Events from the OSD as it prepares operations

- queued_for_pg: the op has been put into the queue for processing by its PG
- reached_pg: the PG has started doing the op
- waiting for *: the op is waiting for some other work to complete before it can proceed (a new OSDMap; for its object target to scrub; for the PG to finish peering; all as specified in the message)
- started: the op has been accepted as something the OSD should actually do (reasons not to do it: failed security/permission checks; out-of-date local state; etc) and is now actually being performed
- waiting for subops from: the op has been sent to replica OSDs

Events from the FileStore

- commit_queued_for_journal_write: the op has been given to the FileStore
- write_thread_in_journal_buffer: the op is in the journal’s buffer and waiting to be persisted (as the next disk write)
- journaled_completion_queued: the op was journaled to disk and its callback queued for invocation

Events from the OSD after stuff has been given to local disk

- op_commit: the op has been committed (ie, written to journal) by the primary OSD
- op_applied: The op has been write()’en to the backing FS (ie, applied in memory but not flushed out to disk) on the primary
- sub_op_applied: op_applied, but for a replica’s “subop”
- sub_op_committed: op_commited, but for a replica’s subop (only for EC pools)
- sub_op_commit_rec/sub_op_apply_rec from <X>: the primary marks this when it hears about the above, but for a particular replica <X>
- commit_sent: we sent a reply back to the client (or primary OSD, for sub ops)

Many of these events are seemingly redundant, but cross important boundaries in the internal code (such as passing data across locks into new threads).

FLAPPING OSDS

We recommend using both a public (front-end) network and a cluster (back-end) network so that you can better meet the capacity requirements of object replication. Another advantage is that you can run a cluster network such that it is not connected to the internet, thereby preventing some denial of service attacks. When OSDs peer and check heartbeats, they use the cluster (back-end) network when it's available. See Monitor/OSD Interaction for details.

However, if the cluster (back-end) network fails or develops significant latency while the public (front-end) network operates optimally, OSDs currently do not handle this situation well. What happens is that OSDs mark each other down on the monitor, while marking themselves up. We call this scenario 'flapping`.

If something is causing OSDs to 'flap' (repeatedly getting marked down and then up again), you can force the monitors to stop the flapping with:

```
ceph osd set noup      # prevent OSDs from getting marked up
ceph osd set nodown    # prevent OSDs from getting marked down
```

These flags are recorded in the osdmap structure:

```
ceph osd dump | grep flags
flags no-up,no-down
```

You can clear the flags with:

```
ceph osd unset noup
ceph osd unset nodown
```

Two other flags are supported, `noin` and `noout`, which prevent booting OSDs from being marked `in` (allocated data) or protect OSDs from eventually being marked `out` (regardless of what the current value for `mon osd down out interval` is).

Note: `noup`, `noout`, and `nodown` are temporary in the sense that once the flags are cleared, the action they were blocking should occur shortly after. The `noin` flag, on the other hand, prevents OSDs from being marked `in` on boot, and any daemons that started while the flag was set will remain that way.