Ceph 性能优化

作者：黄金华

2017/05/10

修订记录

日期	修订版本	修改描述	作者
05/11/2017	V0.1	初始版本	黄金华

1 Ceph 简介

1.1 引言

Ceph 是一款为块存储、文件存储和对象存储提供统一存储方案的开源分布式存储系统，具有高可靠性、高性能和可扩展性特点。同时，Ceph 融入了下述建设性理念[2]:

- 每个组件都能够线性扩展；
- 无任何单点故障；
- 解决方案必须是基于软件的、开源的、适应性强的；
- 每个组件必须尽可能拥有自我管理和自我修复能力；

1.2 存储类型

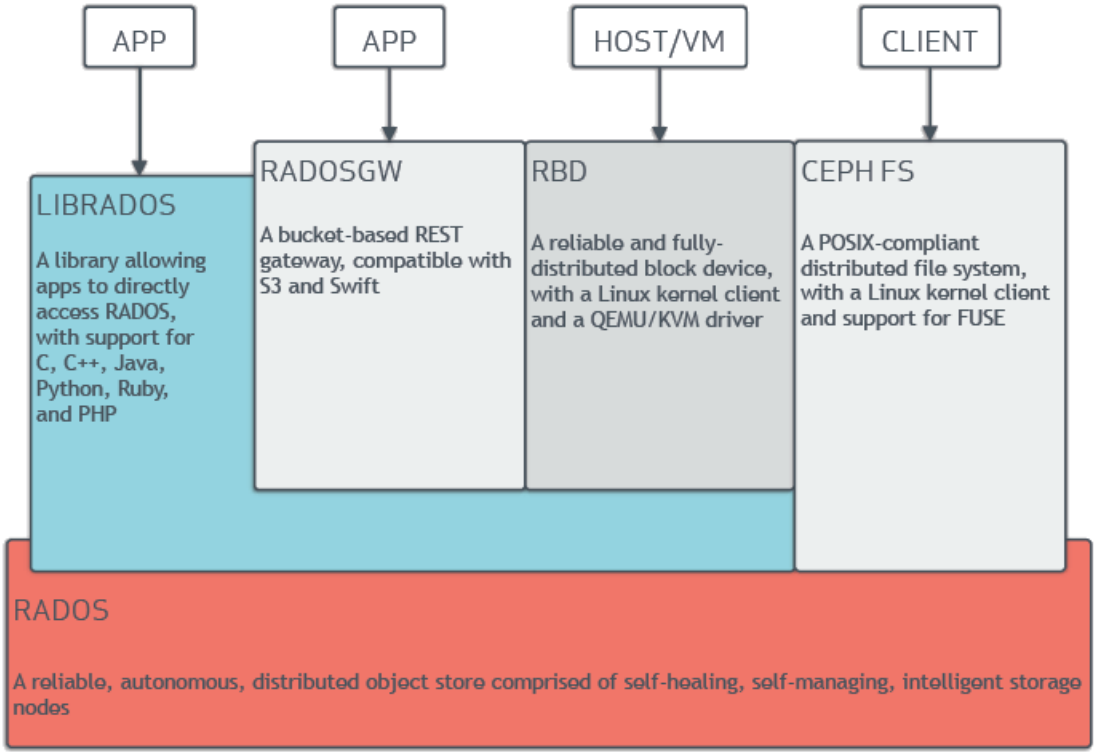
便于后续理解，简单对比几种存储方式以及主要应用场景。

存储类型	主要特点	主要场景
文件存储	● 以文件形式存储数据，并且提供基于“树”型结构的目录和文件名结构 ● 不方便大量文件的检索； ● 文件的大小受到限制；	● 文件共享； ● 本地存档； ● 数据备份和保护；
块存储	● 以“原始块”形式管理和存储数据，并根据“块”地址来唯一确定，同时“块”不再包含元数据信息； ● 方便上层应用直接管理数据；	● 数据库 ● Email Servers ● RAID ● Virtual Machines
对象存储	● 一个对象由数据块和可配置的元数据组成，并且每个对象包含唯一的 ID（扁平的命名空间而非类似目录文件名）； ● 便于扩展； ● 一般提供 RESTful 接口访问；	● 大数据； ● Web Apps; ● Backup Archives;

2 Ceph 内部机制

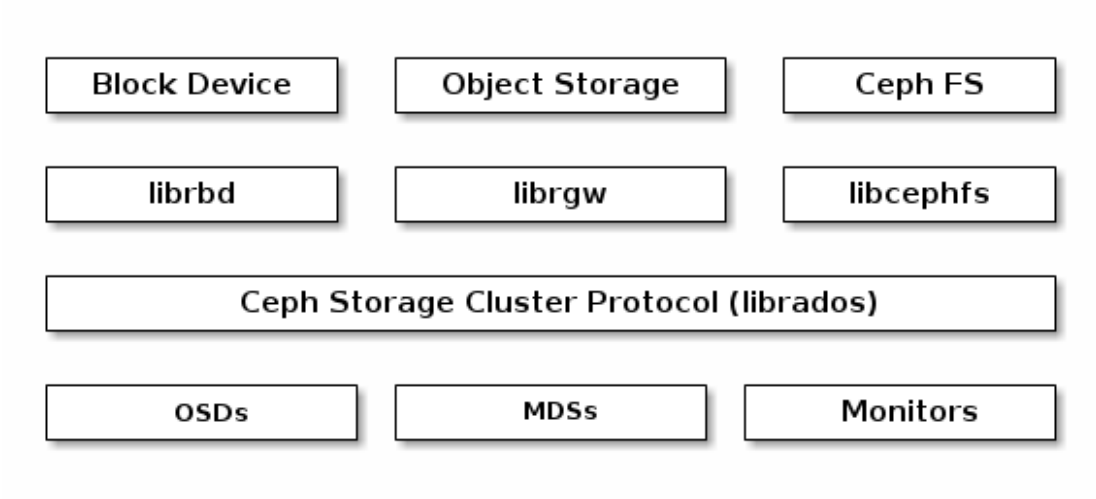
2.1 架构总览

总体上，Ceph 架构如下图：



图表 1 Ceph 架构总览

同时，其主要的组件如下：



图表 2 Ceph 主要组件

其中：

- RADOS(Reliable Autonomic Distributed Object Store) : Ceph 的分布式存储基础框架，并且一切以“对象”形式存储和检索；
- OSD(Object Storage Device): 负责实际数据存储和检索的模块，一般数据会在多个 OSD 上备份实现高可靠性；
- MON(Ceph Monitors) : 负责整个 Ceph 集群的监控，同时启动多个 Mon 并由分布式算法来保证一致性和避免单点故障；
- RBD(Ceph Block Device): 提供块存储接口；

- RGW(RADOS Gateway): 提供 RESTful API 接口，从而实现对象存储和检索；
- CephFS(Ceph File System): 提供兼容 POSIX 接口的分布式文件系统，同时需要启动 MDS(Ceph Metadata Server)，同时 MDS 仅为 CephFS 提供元数据存储和检索；

2.2 可扩展性和高可靠性

与传统的 Client 与中心/Master 节点通信不同的是，Ceph Client 可与存储数据的 OSD 直接通信存储数据。不仅如此， Ceph Client 还会执行一些额外任务尽量（例如计算与哪个 OSD 通信等）。因此，整个 Ceph 系统很容易通过增减 OSD 节点实现线性扩展。

同时，为提高可靠性，每份数据一般会在多个 OSD 备份（一般是一个 Primary OSD 和两个从 OSD）。并且，对于 Monitors 节点，为避免单点故障和提升可靠性，会运行多个 Monitors(一般奇数个)，而且会根据分布式算法 PAXOS 保持多个 Monitors 之间的一致性。

2.3 数据存储流程

2.3.1 存储流程总览

那么 Client 又是怎么知道与哪些 OSD 节点通信呢？又怎样消除中间节点实现高可扩展性和高可靠性？其核心就是 CRUSH(Controlled Replication Under Scalable Hashing，一种伪随机数据分布算法)，主要功能是将输入（一般是对象或对象组 ID）映射到一系列存储设备名（也就是 OSD）。

首先， Ceph 总体操作流程如下图：

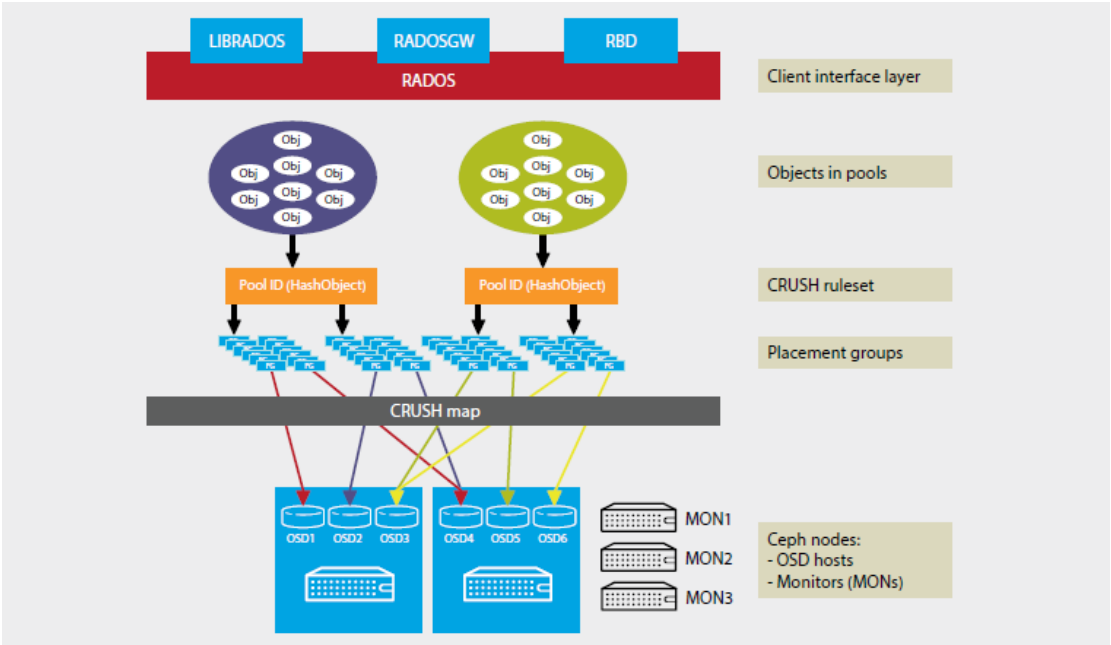
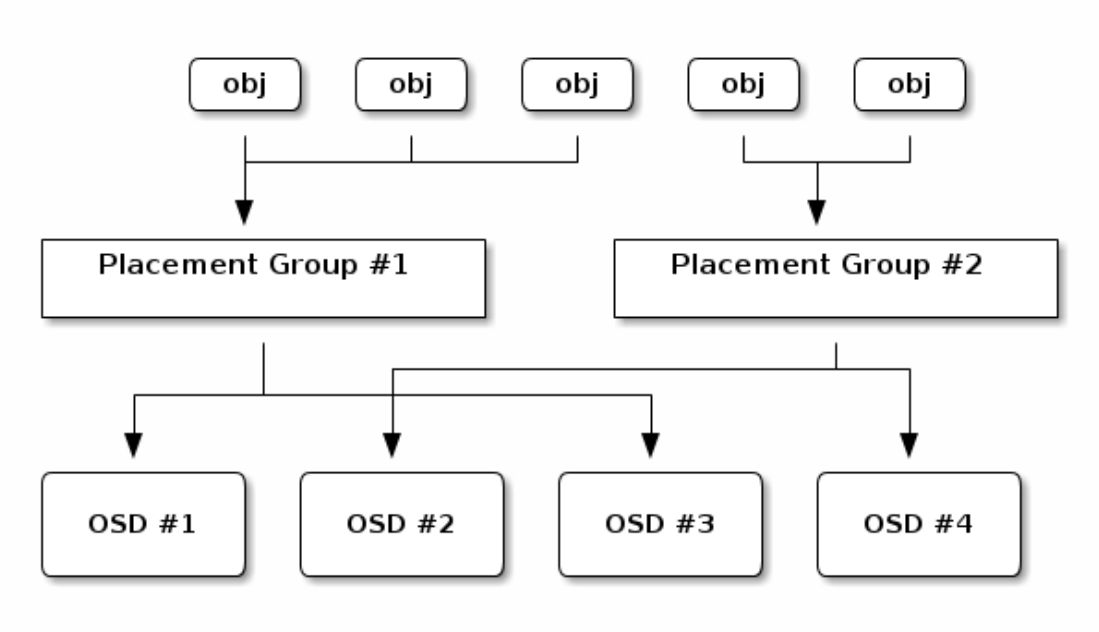


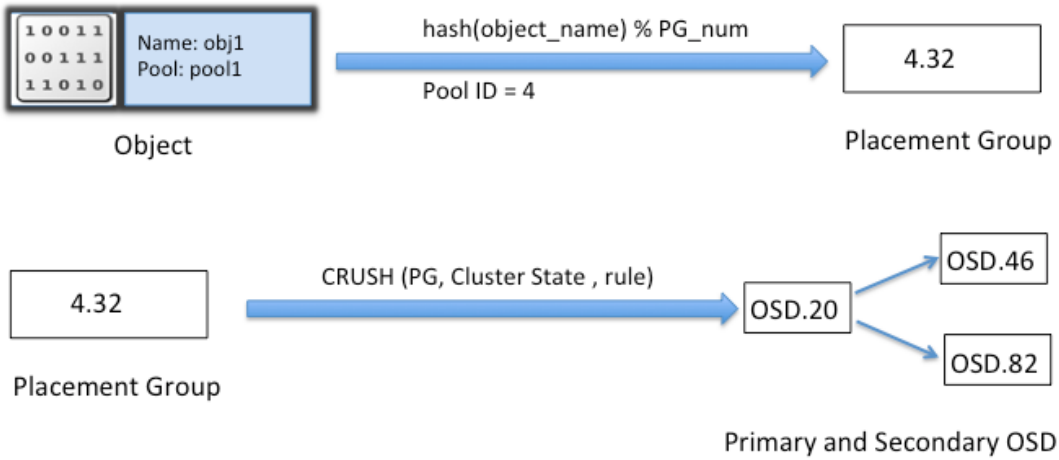
Figure 1. Clients write to Ceph storage pools while the CRUSH ruleset determines how placement groups are distributed across object storage daemons (OSDs).

图表 3 Ceph 存储流程总览

其中，每个对象属于某个存储池（Pool），而若干对象的逻辑集合组成一个 PG(Place Group)。同时根据 Ceph 存储池的副本级别（replication level），每个 PG 被复制到 Ceph 集群中多个 OSD 中（此处假定采用默认的复制机制，而非纠删码机制）。下图 4 是对象与 PG 的逻辑关系图，而图 5 描述如何将对象映射到 OSD 的流程图。



图表 4 对象-PG-OSD 的关系图



图表 5 对象到 OSD 的映射流程图

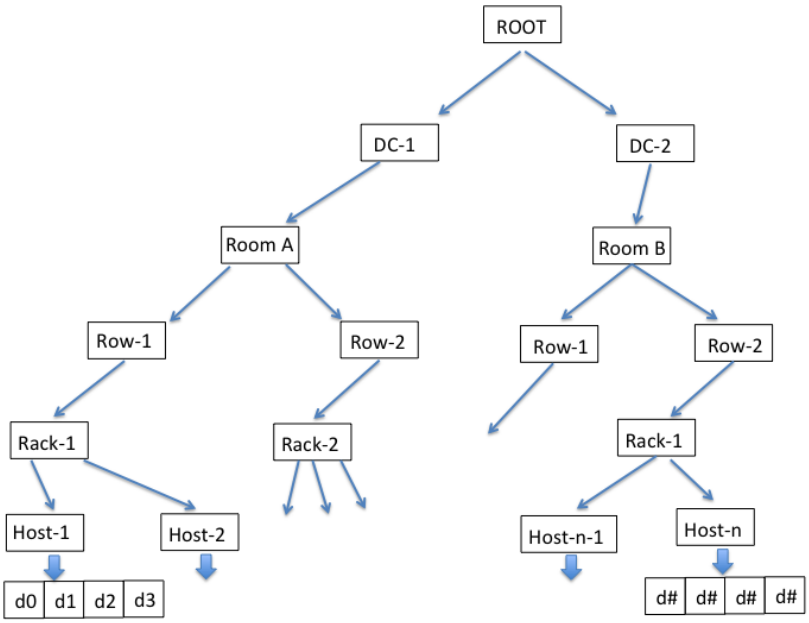
而在此过程中，要获知整个系统的拓扑结构，则依赖 Cluster Map。具体包括：

- Monitor Map：包含每个 Monitor 的 fsid，位置，IP 地址以及端口等信息；
- OSD Map：包含 OSD 列表以及其状态，存储池列表，副本大小，PG 个数等；
- PG Map：包含 PG 版本号、时间戳、OSD Map 版本号、每个 PG 的 ID、对象数量以

及状态等；

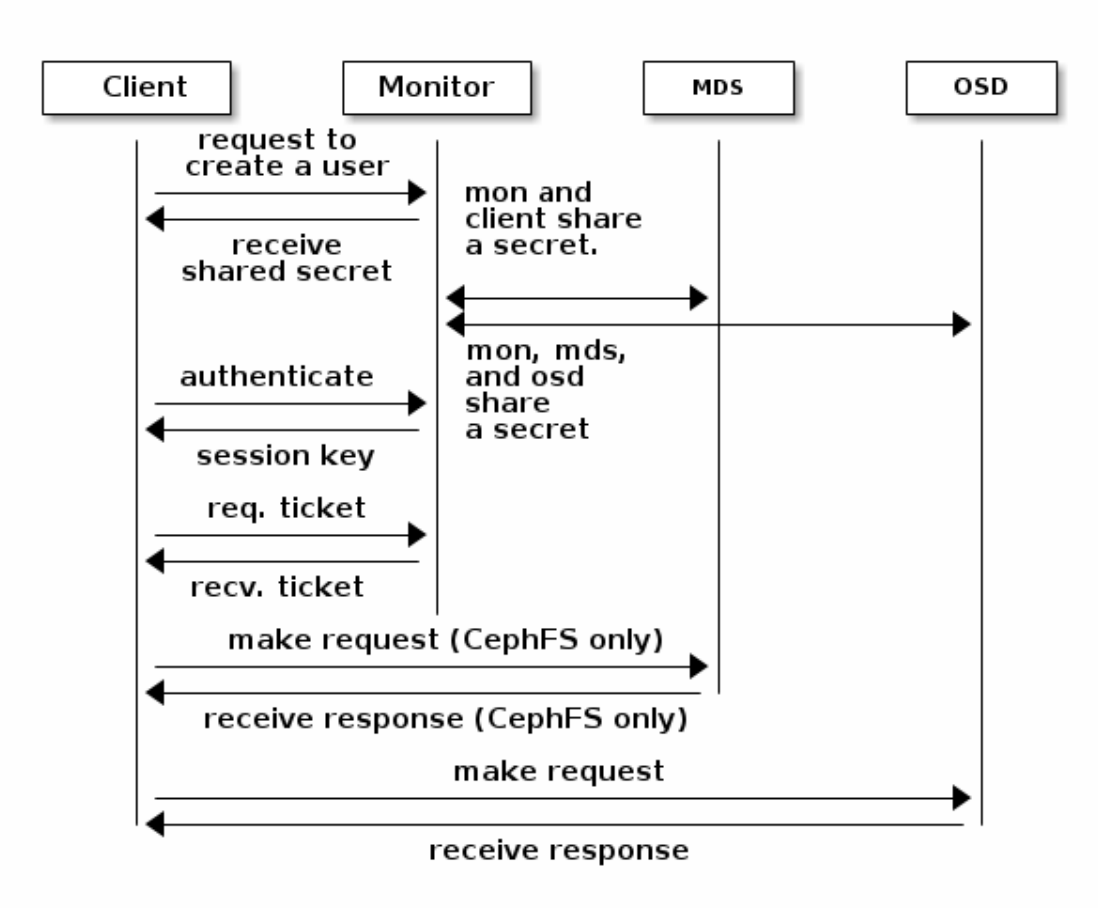
- CRUSH Map：包含集群设备列表、bucket 列表、分层结构以及故障域，主要包含四个部分：
  - ✧ Devices：指定存储设备，通常指 OSD
  - ✧ Bucket Types：定义 Bucket 分层结构；
  - ✧ Bucket Instances：指定 Bucket 实例；
  - ✧ Rules：指定选择 Bucket 的方式；

  下图是 Bucket 层级示例图：



- MDS Map：包含元数据存储池的 ID、集群 MDS 数量以及状态

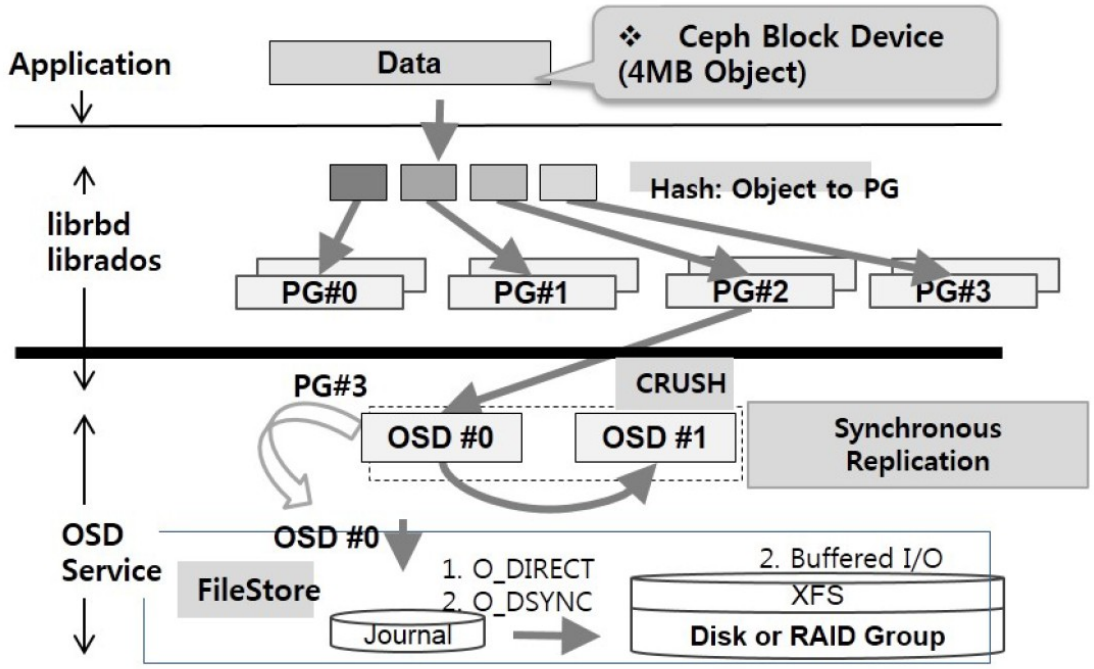
同时，为安全起见，需要一系列认证过程。正如前文可扩展性一节所描述的那样，其中的关键是让 Client 一旦认证后可以与 OSD 直接通信，并且要尽量避免与类似影响扩展性的中心节点交互。其基本认证交互流程如下：



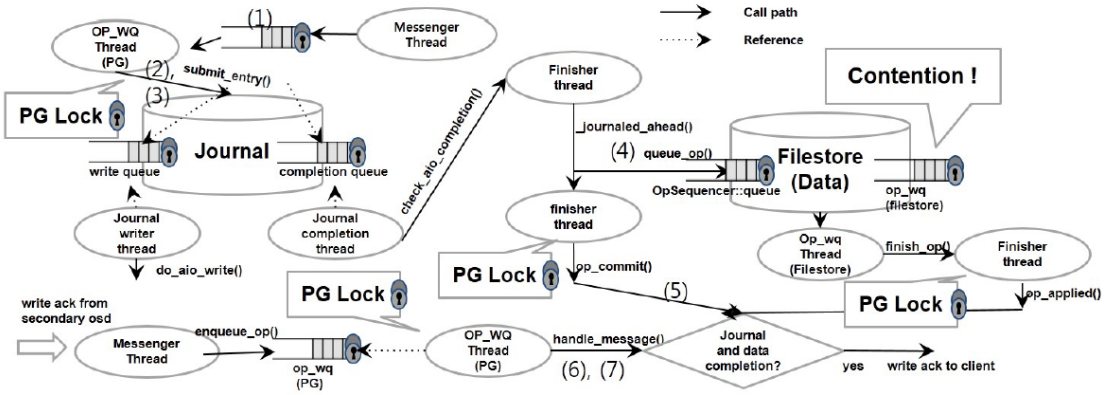
图表 6 Ceph 认证交换流程图

2.3.2 OSD 写数据实例

一个典型的 OSD 数据写流程（含 OSD 内部处理）如下图所示：



a) Structure of Ceph



b) Detailed structure of OSD

图表 7 OSD 内部写流程图

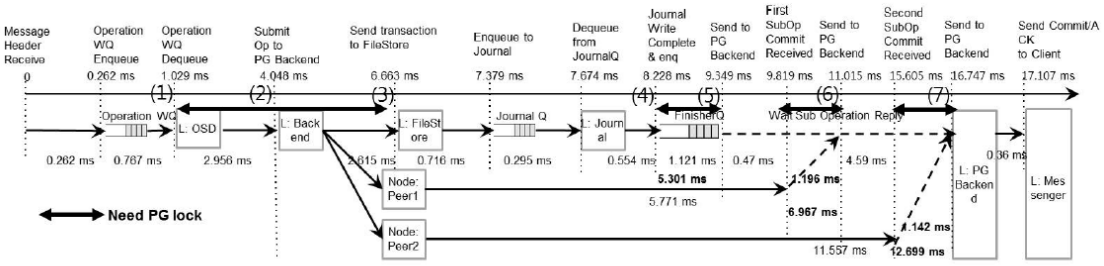


Figure 3: Ceph latency analysis for write path

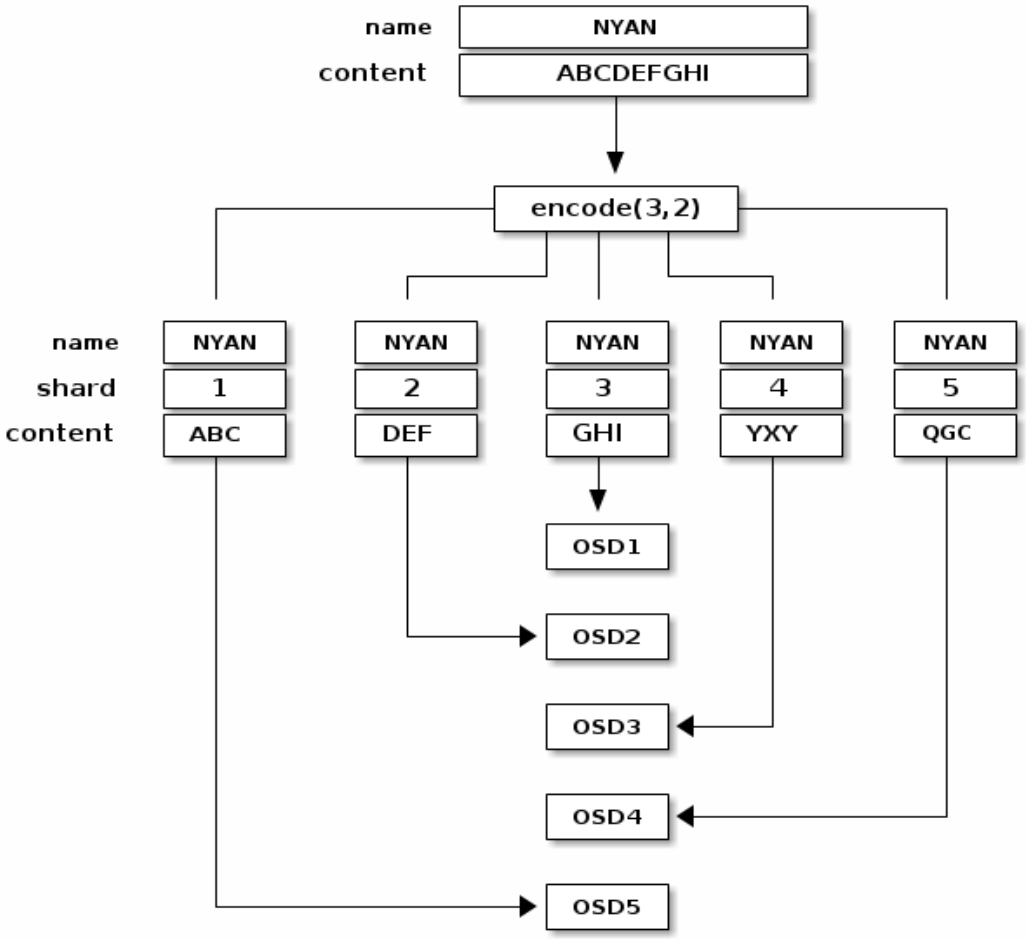
2.4 Ceph 纠删码

与传统通过备份多份提供可靠性不同，纠删码机制通过对原数据切分并添加额编码信息从而提供数据保护和高可靠性。纠删码所占内存和磁盘相对较少，但会引入一定的读写性能开销，因此更适合存储大量读写不频繁的数据。

通常来说，Ceph 纠删码会将每个对象切分成 K+M 个 chunks 保存，其中：

- K 代表数据 chunk 个数；
- M 代表编码 chunk 个数；

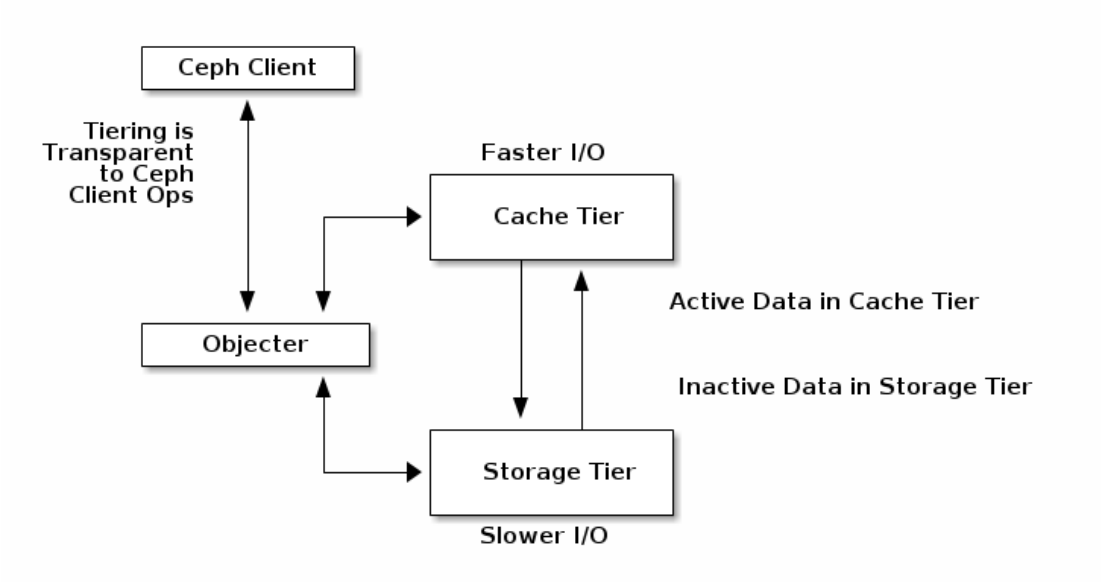
以（K=3, M=2）的纠删码对象存储示意图如下：



图表 8 纠删码对象存储示意图

2.5 缓存分层（Cache Tiering）

通过将一部分数据存储到缓存层，从而给客户提供更好的 I/O 性能。



图表 9 缓存分层示意图

- **Writeback 模式：**
  - ✧ 客户端写数据到缓存分层中则立即会收到确认回复。基于所配置的刷新/删除策略，数据将从缓存迁移到存储层。

- **Read-Only 模式：**
  - ✧ 写请求依旧直接存到后端的存储设备，而只从 Cache Tier 中读取数据（如没有相关数据，则先要从后端存储拷贝到 Cache Tier 中）。

3 性能优化

3.1 基本方法论

对于性能优化的基本方法论，以及基本工具可参阅：https://zhuanlan.zhihu.com/p/25013051

3.2 基准测试

对于 Ceph，其基准测试可包括几部分[10]：

- 基础磁盘和网络的性能：一般借助 dd 测试磁盘基础性能，同时使用 iperf 测试网路基础性能；
  - ✧ 特别的，当测试 SSD 是否适合日志写设备时，可使用如下步骤：
    1) 关闭磁盘 write cache 功能：hdparm –W 0 /dev/sda 0
    2) 关闭 Controller Cache：
       - hpacucli ctrl slot=2 modify dwc=disable

- hpacucli controller slot=2 logicaldrive 1 modify arrayaccelerator=disable
- slot 和 logicaldriver 号需要根据具体情况做相关更改

3) *fio –filename=/dev/sda –direct=1 –sync=1 –rw=write –bs=4k –numjobs=1 –iodepth=1 –runtime=60 –time_based –group_reporting –name=journal-test*

4) 根据测试需要增加 numberjobs

✧ Netcat 也可测试网路带宽，例如：

1) 服务器启动：nc –v –v –l –n 2222 > /dev/null

2) 客户端测试：time dd if=/dev/zero | nc –v –v –n <ipaddress> 2222(port number)

- 单个 OSD 的基础性能：osd tell 可测试单个 OSD 性能；
- RADOS 性能：可借助 rados bench 测试 RADOS；
- Block Device 性能：可借助 FIO 测试块存储性能；

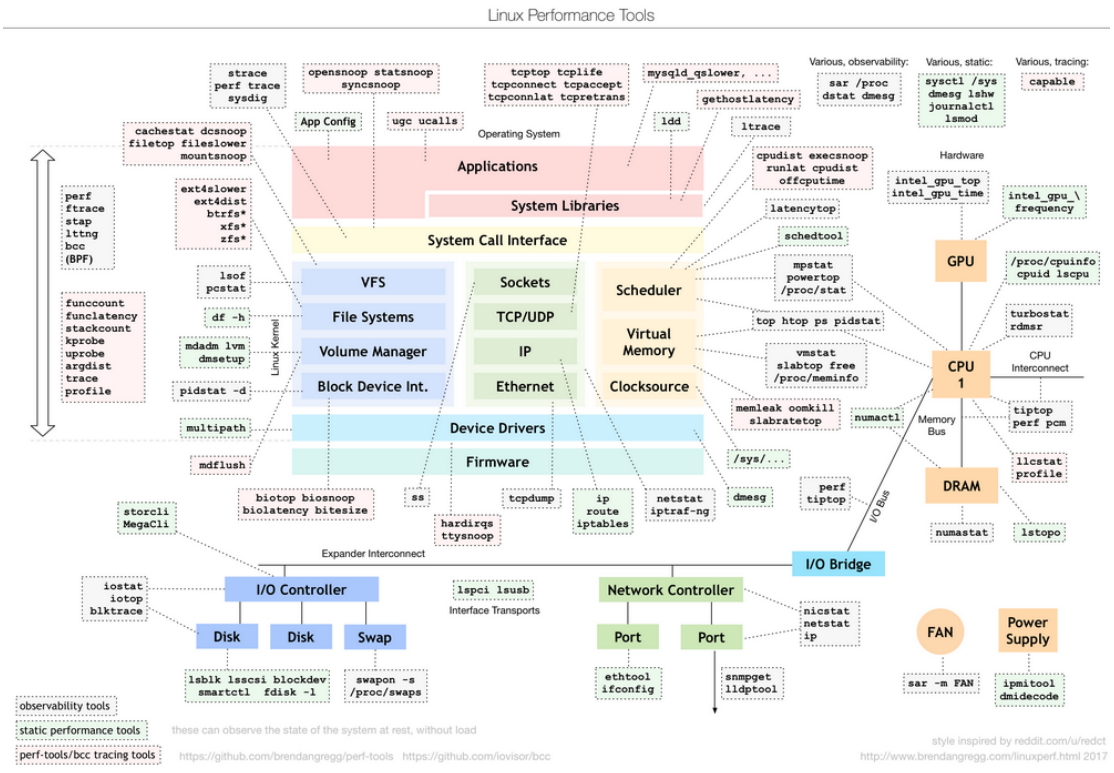
具体测试脚本可参阅：
https://github.com/sjtuhjh/appbenchmark/tree/master/apps/ceph/ceph_test1
或者 https://github.com/ceph/cbt 。

3.3 性能监控

3.3.1 基础性能监控

对 CPU/Memory/Disk/Networking/Kernel 等监控，可采用 perf/ftrace/bcc/top/dstat 等多种工具，既可以支持静动跟踪，也支持动态跟踪。

具体工具可参考 http://github.com/sjtuhjh/perftools，在此不再赘述。



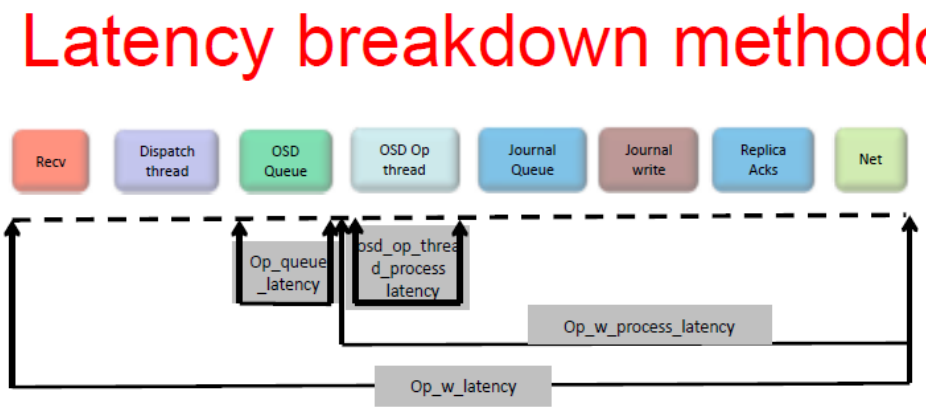
3.3.2 应用内部监控

基础监控：

- ceph health/ceph health detail (checking cluster health)
- ceph –w (watching cluster events)
- ceph df (cluster utilization statistics)
- ceph –s （checking cluster status）
- ceph auth list (cluster authentication keys)
- ceph mon stat/ ceph mon dump (mon status)
- ceph quorum_status(mon quorum status)
- ceph osd tree (osd tree overview)
- ceph osd dump(osd statistics)
- ceph osd crush dump / ceph osd crush rule list/ ceph osd crush rule dump <crush_rule_name> (checking crush map)
- ceph osd find <numeric_osd_id> (find osd location)
- ceph osd perf
- ceph pg stat / ceph pg dump(ceph placegroup statistics)
- ceph pg <pg_id> query / ceph pg dump_stuck <unclean | inactive | stale>
- ceph mds stat/ceph mds dump (monitoring MDS)
- 内存监控：
  - ✧ apt-get/yum install google-perftools
  - ✧ ceph tell osd.0 heap start_profiler
  - ✧ ceph tell osd.0 heap dump
  - ✧ google-pprof –text /usr/bin/ceph-osd /var/log/ceph-osd.0.profile.0001.heap (也可添加—base old.heap 选项与 old.heap 对比内存的变化)
- 5 个主要指标：
  - ✧ health != OK
  - ✧ online moitor nodes doesn’t reach quorum
  - ✧ OSD nodes down but still in (it might fail to recovery). Check ceph.num_osds, ceph.num_in_osds, and ceph.num_up_osds.
  - ✧ reach full capacity (by checking ceph.aggregate_pct_used and ceph.num_full_osds/ceph.num_near_full_osds)
  - ✧ Be on time (NTP，时间抖动不超过 0.05s)

同时，可以从“时延”来统计分析系统瓶颈。例如，可将整个处理流程分成若干阶段，并分析每个阶段的时延。目前可借助下列命令查看 OSD 性能计数器（perf counter），并且可借助 collectd 或 statsd 做进一步分析：

- ceph daemon <osd.0> perf schema
- ceph daemon <osd.o> perf dump



图表 10 Ceph 时延分析法

更多分析工具，可参考 https://github.com/sjtuhjh/perftools/tree/master/apptools/ceph/

3.4 基础优化

3.4.1 BIOS/CPU/Memory

在对应用优化之前，要确保 BIOS/CPU/Memory 配置恰当，否则后续优化往往事倍功半。为此，需要借助一些测试工具来验证硬件功能是否工作正常（或者最优）。值得注意的是，开始测试的时候常常无法根据数据来确定是否工作正常，因此可将数据与历史优化后的数据或者对应的 X86 测试数据进行对比。常见的测试工具如下：

- **CPU**:
  SpecInt2006(https://github.com/sjtuhjh/applications/tree/master/apps/cpu/spec_cpu2006)

- **Memory**: lmbench/Stream
- **DISK : I/O** : fio 测试
  - RAID1：将两块硬盘构成 RAID 磁盘阵列，其容量仅相当于一块磁盘，但另一块磁盘总是保持完整的数据备份。一般支持“热交换”。
  - RAID10：首先建立两个独立的 RAID1，然后将两个独立的 RAID1 再组成一个 RAID0

  建议配备 RAND10，同时支持 BBU(Battery Backup Unit)和 WRITE-BACK 功能的存储设备。这样既可以有良好的读写性能，又能防止意外情况下（例如突然断电）存储内容丢失。

  另一方面，对于存储大量的数据，可以考虑采用 JBOD(把若干物理磁盘组合成一个逻辑磁盘，但并不提供保护和恢复)，但需要其他技术来提供保护和恢复。

  - **Block 缓存**
    - **Bcache**(Linux Kernel 默认支持)：允许将高速率的磁盘如 SSD 作为低速率磁盘（如机械硬盘）的缓存来使用，从而既利用了 SSD 的高速率，同时又利

用了机械硬盘的大容量。

- ✧ **LVM Cache**（dm-cache）：利用在高速率磁盘如 SSD 上建立的逻辑卷（LV）作为在低速率磁盘上建立的逻辑卷的缓存池。

- **BIOS 电源管理**：将电源管理设置成“性能”模式以便最大发挥 CPU 性能；

常见的基准测试工具也可参考 https://github.com/sjtuhjh/applications/tree/master/apps。

3.4.2 OS/Kernel

3.4.2.1文件系统

通常建议采用 EXT4 或 XFS 文件系统（优选 XFS，同时虽然 BTFS 性能也不错但暂不适合生产环境）

- Barriers I/O (Barrier 之前的所有 I/O 请求必须在 Barrier 之前结束，并持久化到非易失性介质中。而 Barrier 之后的 I/O 必须在 Barrier 之后执行。如果本身存储设备自带电池足以预防在突然掉电时其缓存内容依然可以被正确持久化，则可以关闭 Barrier 一般提高性能)。一般文件系统挂载时提供选项可以关闭 barriers,例如：mount –o nobarrier /dev/sda /u01/data
- 打开 noatime，减少不必要的 I/O 操作，例如：mount –o nobarrier,noatime …
- 调整 vm.overcommit (将其设置为 2，尽量避免 OOM(Out of Memory killer))
- 写缓存优化（需要根据需要调整参数，当脏页刷写太频繁，会导致过多 I/O 请求，而刷写频率过低，又可能导致所占内存过多，而在不得不刷写脏页时导致应用性能波动）：
  - vm.dirty_background_ratio: 当脏页数量达到一定比例时，触发内核线程异步将脏页写入磁盘；
  - vm.dirty_ratio: 当脏页数量达到一定比例时，系统不得不将脏页写入缓存。此过程可能导致应用的 I/O 被阻塞。
- I/O 调度
  - 对于普通磁盘，建议使用 deadline 调度方式，而固态硬盘各种调度方式影响并不大；
- Read-ahead 调整：
  - blockdev –getra /dev/sda 查看 sda
  - blockdev –setra <sectors> /dev/sda （设置预读缓冲大小）
- vm.swappiness=0 （尽量避免使用 SWAP）
- Linux 文件系统 inode 数评估：事先需要评估文件大小和文件个数，从而确定 inode 个数，避免 inode 数不足。

3.4.2.2NUMA & CGroup

一般建议关闭 NUMA（或者使用交织调度并关闭 NUMA BALANCING）性能会更好。主要原因

是当 NUMA 节点内存分配不足时，内核可能采取”reclaim_mode”从而导致 cache 命中率下降最终导致影响系统性能。

因此，一般也建议将“vm.zone_reclaim_mode”设置为 0（意味着当本节点内存不足时，优先从其他节点分配内存，而非从本节点分配）。

此外，可以通过 CGroup 把 OSD 绑定到特定 CPU 上可防止 OSD 在不同 CPU 上的频繁切换。具体脚本可参考 Pincpus 脚本。

3.4.2.3资源限制

增大 kernel pid max，例如：echo 4194303 > /proc/sys/kernel/pid_max。

3.4.2.4存储设备设置优化

3.4.2.4.1 分区对齐

分区对齐的目的是让逻辑块与物理块对齐从而减少 I/O 操作。通常可以借助 parted 命令来自动对齐，例如：

device=/dev/sdb

parted –s –a optimal $device mklable gpt

parted –s –a optimal $device mkpart primary ext4 0% 100%

其中 optimal 表示要尽量分区对齐从而达到最好性能。

3.4.2.4.2 SSD 优化

SSD 支持 TRIM 功能，打开该功能可以防止未来读写性能恶化。通过 Btrfs, EXT4, JFS 和 XFS 文件系统都支持该功能。可通过下列命令进行配置：

- 普通磁盘: 在 mount 的时候添加 discard 选项，例如：
  /dev/sda / ext4 rw,discard 0 1
- LVM (Logical Volumes)： 在/etc/lvm/lvm.conf 中添加 issue_discards=1；

3.4.2.4.3 I/O 调度

- 一般建议将/sys/block/<设备名>/queue/scheduler 设置为 deadline（普通 SATA）或 noop（SSD 等）。尤其是对普通机械硬盘，不要将其设置为 cfq 算法。
- 适当增大 /sys/block/<设备名>/queue/nr_requests

3.4.2.4.4 SWAP

尽量禁止使用 SWAP，从而避免性能波动。可在/etc/sysctl.conf 中添加 vm.swappiness=0 并通过 sysctl –p 使其生效。

3.4.2.4.5 Readahead

使用 blockdev –setra <value> 设置为较大的值（例如 8192）

3.4.3 网络

- 关闭 Nagle 算法（当存在大量小请求时）；
- ifconfig <interface> mtu 9000 (enabling Jumbo frames);
- 网络相关参数配置：

```
## Increase Linux autotuning TCP buffer limits
## Set max to 16MB (16777216) for 1GE
## 32MB (33554432) or 54MB (56623104) for 10GE

# 1GE/16MB (16777216)
#net.core.rmem_max = 16777216
#net.core.wmem_max = 16777216
#net.core.rmem_default = 16777216
#net.core.wmem_default = 16777216
#net.core.optmem_max = 40960
#net.ipv4.tcp_rmem = 4096 87380 16777216
#net.ipv4.tcp_wmem = 4096 65536 16777216

# 10GE/32MB (33554432)
#net.core.rmem_max = 33554432
#net.core.wmem_max = 33554432
#net.core.rmem_default = 33554432
#net.core.wmem_default = 33554432
#net.core.optmem_max = 40960
#net.ipv4.tcp_rmem = 4096 87380 33554432
#net.ipv4.tcp_wmem = 4096 65536 33554432

# 10GB/54MB (56623104)
net.core.rmem_max = 56623104
net.core.wmem_max = 56623104
net.core.rmem_default = 56623104
```

net.core.wmem_default = 56623104
net.core.optmem_max = 40960
net.ipv4.tcp_rmem = 4096 87380 56623104
net.ipv4.tcp_wmem = 4096 65536 56623104

Increase number of incoming connections. The value can be raised to bursts of request, default is 128
net.core.somaxconn = 1024

Increase number of incoming connections backlog, default is 1000
net.core.netdev_max_backlog = 50000

Maximum number of remembered connection requests, default is 128
net.ipv4.tcp_max_syn_backlog = 30000

Increase the tcp-time-wait buckets pool size to prevent simple DOS attacks, default is 8192
net.ipv4.tcp_max_tw_buckets = 2000000

 # Recycle and Reuse TIME_WAIT sockets faster, default is 0 for both
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1

Decrease TIME_WAIT seconds, default is 30 seconds
net.ipv4.tcp_fin_timeout = 10

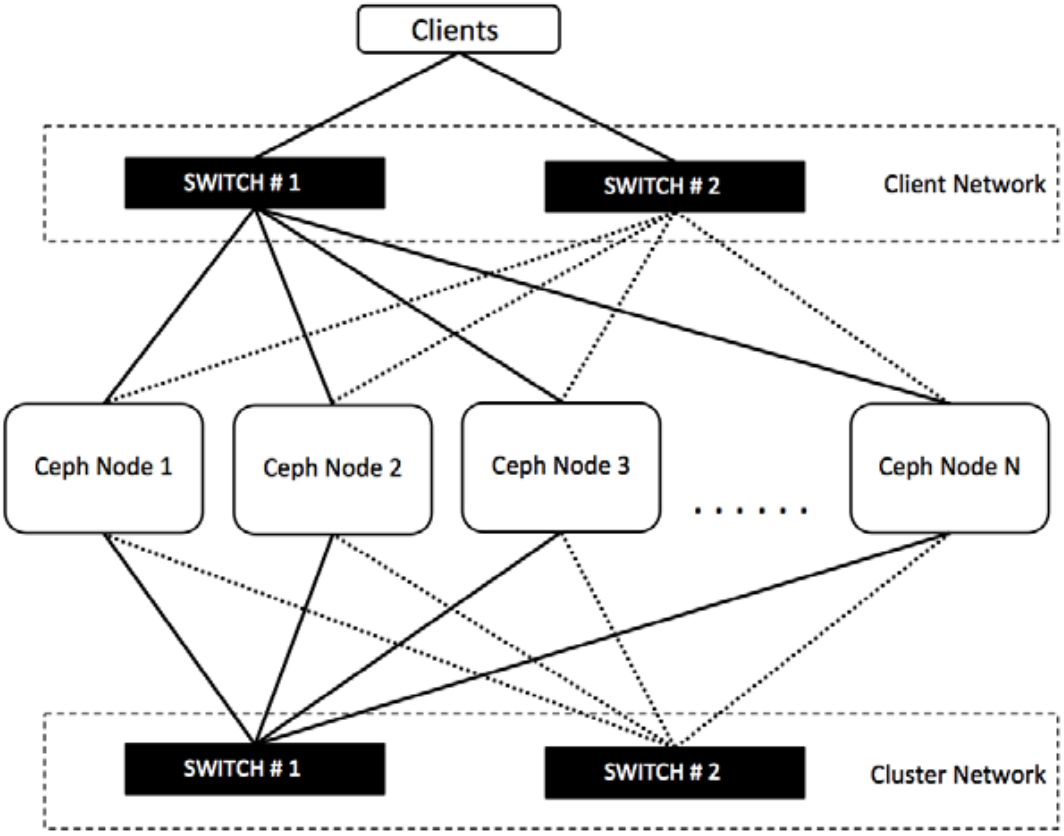
Tells the system whether it should start at the default window size only for TCP connections
that have been idle for too long, default is 1
net.ipv4.tcp_slow_start_after_idle = 0

#If your servers talk UDP, also up these limits, default is 4096
net.ipv4.udp_rmem_min = 8192
net.ipv4.udp_wmem_min = 8192

Disable source redirects
Default is 1
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.all.accept_redirects = 0

Disable source routing, default is 0
net.ipv4.conf.all.accept_source_route = 0

- 使用 **RDMA (Remote Direct Memory Access)（需要对应的网卡支持，如 Mellanox 的 Connect-X 系列）**，具有以下优点：
  - ✧ 零拷贝：数据拷贝直接在两端缓存中执行，而无需再网络软件协议栈之间拷贝；
  - ✧ Kernel bypass: 应用可以直接从用户态拷贝数据，而无需内核干预；
  - ✧ No CPU Involvement: 不影响（或者中断）现有 CPU 的运行；
- 一般网络测试布局如下：



图表 11 网路测试拓扑结构图

3.5 Ceph 优化

总体上，根据不同的优化目标（如 IOPS 或吞吐量等），其典型的 Ceph 的优化示例如下图所示[4]：

OPTIMIZATION CRITERIA	PROPERTIES	EXAMPLE USES
IOPS-OPTIMIZED	• Lowest cost per IOPS • Highest IOPS • Meets minimum fault domain recommendation (single server is less than or equal to 10% of the cluster)	• Typically block storage • 3x replication (HDD) or 2x replication (SSD) • MySQL on OpenStack clouds
THROUGHPUT-OPTIMIZED	• Lowest cost per given unit of throughput • Highest throughput • Highest throughput per BTU • Highest throughput per watt • Meets minimum fault domain recommendation (single server is less than or equal to 10% of the cluster)	• Block or object storage • 3x replication • Active performance storage for video, audio, and images • Streaming media
CAPACITY-OPTIMIZED	• Lowest cost per TB • Lowest BTU per TB • Lowest watt per TB • Meets minimum fault domain recommendation (single server is less than or equal to 15% of the cluster)	• Typically object storage • Erasure coding common for maximizing usable capacity • Object archive • Video, audio, and image object archive repositories

图表 12 Ceph 优化示例

详细的优化在下述章节中描述。

3.5.1 Ceph Cluster 配置优化

- Global Config
  - max open files = 131072
  - filestore xattr use omap = true (Extended attributes, 用来存储元数据)
  - filestore min sync interval = 10
  - filestore max sync interval = 15
  - **FileStore Queue**
    - filestore queue max ops = 25000
    - filestore queue max bytes = 10485760
    - filestore queue committing max ops = 5000
    - filestore queue committing max bytes = 10485760000
    - filestore op threads = 32
  - **OSD Journal Tuning**
    - journal max write bytes = 1073714824
    - journal max write entries = 10000 (maximum number of entries the journal can write at once)
    - journal queue max ops = 50000
    - journal queue max bytes = 10485760000
  - **OSD Config tuning**
    - osd max write size = 512
    - osd client message size cap = 2048 (maximum size of client data)

        - osd deep scrub stride = 131072 (the size in bytes that is read by OSDs when doing a deep scrub)
        - osd op threads = 16 (Ceph thread numbers)
        - osd disk threads = 4(number of disk threads to perform OSD-intensive operations such as recovery and scrubbing)
        - osd map cache size = 1024 (OSD map cache in megabytes)
        - osd map cache b1 size = 128 (size of OSD map caches stored in-memory in megabytes)
        - osd mount options xfs =”rw, noatime, inode64, logbsize=256k, delaylog, allocsize =4M”
    - **OSD Recovery tuning**
        - osd recovery op priority = 4 (lower the number, higher the recovery priority. Higher recovery priority might cause performance degradation)
        - osd recovery max active = 10 (maximum number of active recover requests)
        - osd recovery max backfills = 4 (maximum number of backfill operations allowed to/from OSD. The higher the number, the quicker the recovery, which might impact overall cluster performance until recovery finishes)
- OSD Section
    - osd mkfs type = xfs
    - osd journal = /another disk/osd-$cluster-$id/journal [注释：为 journal 采用单独磁盘]
    - osd journal size = 5120
        - `osd journal size = {2 * (expected throughput * filestore max sync interval)}`
        - 因此，需要根据磁盘的带宽（MB/s）与 sync interval 来共同确定 osd journal size 的大小
- MDS Section
    - mds cache size = <>
- Client section
    - rbd cache = true
    - rbd cache size = 268435456 (RBD cache size in bytes)
    - rbd cache max dirty = 134217728 (dirty limit in bytes; after this specified limit, data will be flushed to the backing store)
    - rbd cache max dirty age = 5 (number of seconds during which dirty data will be stored on cache before it’s flushed to the backing store)

3.5.2 PG Number 配置

PG 和 PGP 数量需要根据 OSD 个数进行调整，一般根据以下规则计算：

*Total PGs = (Total number of OSD * 100) / max_replication_count*

3.5.3 CRUSH Map 配置

CRUSH Map 需要根据具体的部署来确定，同时根据“copyset[11]”来提升系统可靠性。

3.5.4 Malloc 机制

使用 jemalloc （编译的时候指定：--with-jemalloc –without-tcmalloc）。
同时，如使用 tcmalloc，需要适当增大 cache 的大小，如下：
TCMALLOC_MAX_TOTAL_THREAD_CACHE_BYTES=268435456 //256MB

3.6 算法代码优化

与具体 CPU 相关的优化，待补充。

4 引用

[1] www.Ceph.com
[2] Ceph Cookbook
[3] Viju, Optimizing Ceph for All-Flash Architectures
[4] Red Hat Ceph Storage Hardware Configuration Guide
[5] Myoungwon Oh, Performance Optimization for All Flash Scale-out Storage
[6] Jian Zhang, Best Practices for Increasing Ceph Performance with SSD
[7] www.storagecraft.com/blog/storage-wars-file-block-object-storage
[8] Sage A. Weil, Ceph: A Scalable, High-Performance Distributed File System
[9] Sage A. Weil, CRUSH: Controlled, Scalable , Decentralized Placement of Replicated Data
[10]Diana Gudu, Evaluating the Performance and Scalability of the Ceph Distributed Storage System
[11] Asaf Cidon, Copysets: Reducing the Frequency of Data Loss in Cloud Storage