开源 CEPH 存储系统性能调参优化

创建作者：侯超
创建时间：2025/01/12

一、CEPH 架构及性能优化

IT 领域无非就是计算、存储、网络，不管是哪一种系统，本质就是一个“盒子”，硬件加持，软件驱动，可以从硬件和软件两个层面进行分析。
硬件层面：
硬件规划设置
磁盘类型
BIOS 设置(固件或者驱动层)
软件层面：
LINUX 内核
Ceph 存储系统配置参数(在下个小结详细介绍)

1.1 硬件层面

1.1.1 CPU

Ceph-osd 进程在运行过程中会消耗 cpu 资源，一般会为每一个 ceph-osd 进程绑定到一个 cpu 核上， 就是所谓的绑核。一般来说，只有在运营商公开招标存储产品集采 POC 性能测试时， 为提高自家产品在各项测试指标的性能数据， 比友商好时，进行固定的绑核，或者特定场景或者 IO 模型进行绑核，正式交付场景还是要慎用。Ceph-mon 进程不会吃太多 cpu 资源，不用考虑为 ceph-mon 预留更多的 CPU 资源，另外针对文件存储 ceph-mds 也需要更多的 CPU 资源。另外具体的绑核操作可以另外介绍，以及相关的使用，脚本等等，注意，绑核策略个人感觉不太适合讯飞云主机的相关业务。

1.1.2 内存

Ceph-mon 和 ceph-mds 需要 2G 内存，每一个 ceph-osd 进程需要 1G 内存。这篇文章里是这样讲的，实际使用可能会多一些，比如 3G-4G。可以调一下根据具体情况来定。
Ceph 相关 osd 内存限制操作如下:
永久修改：
cd /etc/ceph/ceph.conf
在[osd]一栏，添加如下:
osd_memory_target = 4294967296
重启 osd 服务, systemctl restart ceph-osd.x.service
注意，存储集群要一个 osd 一个 osd 重启，防止一起重启 osd 服务把集群搞挂。
临时修改:
#ceph tell osd.* injectargs '-- osd_memory_target=3221225472'
或者执行命令: ceph daemon osd.x confit set osd_memory_target 3221225472
参考链接: https://www.cnblogs.com/gustabm/p/15822744.html

1.1.3 网络

万兆网络现在基本上是跑 Ceph 必备的，网络规划上，也尽量考虑分离 cilent 和 cluster 网络。网络接口上可以使用 bond 来提供高可用或负载均衡。

1.1.4 磁盘

使用 SSD 高速介质为存储系统 HDD 盘进行加速。对于 ceph 这种第一代存储架构来说，采用高速介质进行缓存加速是业界主流的玩法。

1.1.5 BIOS

a、 开启 VT 和 HT，VH 是虚拟化云平台必备的，HT 是开启超线程单个处理器都能使用线程级并行计算。
b、 关闭节能设置，可有一定的性能提升。
c、NUMA 思路就是将内存和 CPU 分割为多个区域，每个区域叫做 NODE,然后将 NODE 高速互联。 node 内 cpu 与内存访问速度快于访问其他 node 的内存， NUMA 可能会在某些情况下影响 ceph-osd 。解决的方案，一种是通过 BIOS 关闭 NUMA，另外一种就是通过 cgroup 将 ceph-osd 进程与某一个 CPU Core 以及同一 NODE 下的内存进行绑定。但是第二种看起来更麻烦，所以一般部署的时候可以在系统层面关闭 NUMA。CentOS 系统下，通过修改/etc/grub.conf 文件，添加 numa=off 来关闭 NUMA。这个说的是 BIOS 层面的 numa。

1.2 软件层面

1.2.1 kernel pid max

echo 4194303 > /proc/sys/kernel/pid_max　　#系统允许最大进程数
对于开源 ceph 存储系统来说可以先不管。

1.2.2 设置 MTU，交换机端需要支持该功能，系统网卡设置才有效果

配置文件追加 MTU=9000
对于开源 ceph 存储系统来说可以先不管， 这个需要网络交换机支持才可以，客户现网调优再配置也不迟，或者交付客户部署环境要求支持即可。

1.2.3 read_ahead, 通过数据预读并且记载到随机访问内存方式提高磁盘读操作

echo "8192" > /sys/block/sda/queue/read_ahead_kb
针对块存储场景可以试一下，但是可能会使得客户端内存增大，这个暂时慎用。

1.2.4 swappiness, 主要控制系统对 swap 的使用

echo "vm.swappiness = 0"/etc/sysctl.conf ;　sysctl –p
一般推荐关闭，建议设为 0.

1.2.5 I/O Scheduler，SSD 要用 noop，SATA/SAS 使用 deadline

echo "deadline" >/sys/block/sd[x]/queue/scheduler
默认调度就是这样， 可以不管。

二、CEPH 主要配置参数详解

2.1 概述

Ceph 的配置参数很多，从网上也能搜索到一大批的调优参数，但这些参数为什么这么设置？设置为这样是否合理？解释的并不多。
Ceph 存储系统的调优参数大致分为这么几类：
(1) Throttle 相关的参数;
(2) 线程相关的参数;
(3) 缓存相关的参数;
本文从当前我们的 ceph.conf 文件入手，解释其中的每一项配置，做为以后参数调优和新人学习的依据；

2.2 参数详解

2.2.1 一些固定配置参数

```
1　fsid = 6d529c3d-5745-4fa5-be5f-3962a8e8687c
2　mon_initial_members = mon1, mon2, mon3
3　mon_host = 10.10.40.67,10.10.40.68,10.10.40.69
```

以上通常是通过 ceph-deploy 生成的，都是 ceph monitor 相关的参数，不用修改；

2.2.2 网络配置参数

```
1　public_network = 10.10.40.0/24　　默认值 ""
2　cluster_network = 10.10.41.0/24　默认值 ""
```

public network：monitor 与 osd，client 与 monitor，client 与 osd 通信的网络，最好配置为带宽较高的万兆网络；
cluster network：OSD 之间通信的网络，一般配置为带宽较高的万兆网络；
参考：
http://docs.ceph.com/docs/master/rados/configuration/network-config-ref/

2.2.3 pool size 配置参数

```
1 osd_pool_default_size = 3　　默认值 3
2 osd_pool_default_min_size = 1　默认值 0 // 0 means no specific default; ceph will use size-size/2
```

这两个是创建 ceph pool 的时候的默认 size 参数，一般配置为 3 和 1，3 副本能足够保证数据的可靠性；

2.2.4 认证配置参数

```
1  auth_service_required = none       默认值  "cephx"
2  auth_client_required = none        默认值  "cephx, none"
3  auth_cluster_required = none     默认值  "cephx"
```

以上是 Ceph authentication 的配置参数，默认值为开启 ceph 认证；
在内部使用的 ceph 集群中一般配置为 none，即不使用认证，这样能适当加快 ceph 集群访问速度；

2.2.5 osd down out 配置参数

```
1  mon_osd_down_out_interval = 864000   默认值  300 // seconds
2  mon_osd_min_down_reporters = 2         默认值  2
3  mon_osd_report_timeout = 900           默认值  900
4  osd_heartbeat_interval = 15          默认值  6
5  osd_heartbeat_grace = 60               默认值  20
```

mon_osd_down_out_interval：ceph 标记一个 osd 为 down and out 的最大时间间隔
mon_osd_min_down_reporters：mon 标记一个 osd 为 down 的最小 reporters 个数（报告该 osd 为 down 的其他 osd 为一个 reporter）
mon_osd_report_timeout：mon 标记一个 osd 为 down 的最长等待时间
osd_heartbeat_interval：osd 发送 heartbeat 给其他 osd 的间隔时间（同一 PG 之间的 osd 才会有 heartbeat）
osd_heartbeat_grace：osd 报告其他 osd 为 down 的最大时间间隔，grace 调大，也有副作用，如果某个 osd 异常退出，等待其他 osd 上报的时间必须为 grace，在这段时间段内，这个 osd 负责的 pg 的 io 会 hang 住，所以尽量不要将 grace 调的太大。
基于实际情况合理配置上述参数，能减少或及时发现 osd 变为 down（降低 IO hang 住的时间和概率），延长 osd 变为 down and out 的时间（防止网络抖动造成的数据 recovery）；
参考：
http://docs.ceph.com/docs/master/rados/configuration/mon-osd-interaction/
http://blog.wjin.org/posts/ceph-osd-heartbeat.html

2.2.6 objecter 配置参数

```
1  objecter_inflight_ops = 10240                 默认值  1024
2  objecter_inflight_op_bytes = 1048576000       默认值  100M
```

osd client 端 objecter 的 throttle 配置，它的配置会影响 librbd，RGW 端的性能；
配置建议：
调大这两个值，这两个值可以试一下， 并适当调大看看。

2.2.7 ceph rgw 配置参数

```
1    rgw_frontends = "civetweb port=10080 num_threads=2000"   默认值  "fastcgi, civetweb port=7480"
2    rgw_thread_pool_size = 512                                 默认值  100
3    rgw_override_bucket_index_max_shards = 20                  默认值  0
4
5    rgw_max_chunk_size = 1048576                                 默认值  512 * 1024
6    rgw_cache_lru_size = 1000000                               默认值  10000 // num of entries in rgw cache
7    rgw_bucket_default_quota_max_objects = 10000000            默认值  -1 // number of objects allowed
8
9    rgw_dns_name = object-storage.ffan.com                       默认值
10   rgw_swift_url = http://object-storage.ffan.com             默认值
```

rgw_frontends：rgw 的前端配置，一般配置为使用轻量级的 civetweb；prot 为访问 rgw 的端口，根据实际情况配置；num_threads 为 civetweb 的线程数；
rgw_thread_pool_size：rgw 前端 web 的线程数，与 rgw_frontends 中的 num_threads 含义一致，但 num_threads 优于 rgw_thread_pool_size 的配置，两个只需要配置一个即可；
rgw_override_bucket_index_max_shards：rgw bucket index object 的最大 shards 数，增大这个值能减少 bucket index object 的访问时间，但也会加大

bucket 的 ls 时间；
rgw_max_chunk_size：rgw 最大 chunk size，针对大文件的对象存储场景可以把这个值调大；
rgw_cache_lru_size：rgw 的 lru cache size，对于读较多的应用场景，调大这个值能加快 rgw 的响应速度；
rgw_bucket_default_quota_max_objects：配合该参数限制一个 bucket 的最大 objects 个数；
参考：
http://docs.ceph.com/docs/jewel/install/install-ceph-gateway/
http://ceph-users.ceph.narkive.com/mdB90g7R/rgw-increase-the-first-chunk-size
https://access.redhat.com/solutions/2122231

2.2.8 debug 配置参数

```
1       debug_lockdep = 0/0
2       debug_context = 0/0
3       debug_crush = 0/0
4       debug_buffer = 0/0
5       debug_timer = 0/0
6       debug_filer = 0/0
7       debug_objecter = 0/0
8       debug_rados = 0/0
9       debug_rbd = 0/0
10      debug_journaler = 0/0
11      debug_objectcatcher = 0/0
12      debug_client = 0/0
13      debug_osd = 0/0
14      debug_optracker = 0/0
15      debug_objclass = 0/0
16      debug_filestore = 0/0
17      debug_journal = 0/0
18      debug_ms = 0/0
19      debug_mon = 0/0
20      debug_monc = 0/0
21      debug_tp = 0/0
22      debug_auth = 0/0
23      debug_finisher = 0/0
24      debug_heartbeatmap = 0/0
25      debug_perfcounter = 0/0
26      debug_asok = 0/0
27      debug_throttle = 0/0
28      debug_paxos = 0/0
29      debug_rgw = 0/0
```

关闭了所有的 debug 信息，能一定程度加快 ceph 集群速度，但也会丢失一些关键 log，出问题的时候不好分析；
参考：
http://www.10tiao.com/html/362/201609/2654062487/1.html
建议在讯飞业务中部署块存储产品时关闭所有业务 debug 日志。

2.2.9 osd op 配置参数

```
1   osd_enable_op_tracker = false          默认值 true
2   osd_num_op_tracker_shard = 32          默认值 32
3   osd_op_threads = 10                    默认值 2   #不重要，暂时不用管
4   osd_disk_threads = 1                   默认值 1
5   osd_op_num_shards = 32                      默认值 5 #网上有人做过实验，性能提升不明显
6   osd_op_num_threads_per_shard = 2       默认值 2
```

osd_enable_op_tracker：追踪 osd op 状态的配置参数，默认为 true；不建议关闭，关闭后 osd 的 slow_request，ops_in_flight，historic_ops 无法正常统计；

```
1 # ceph daemon /var/run/ceph/ceph-osd.0.asok dump_ops_in_flight
2 op_tracker tracking is not enabled now, so no ops are tracked currently, even those get stuck.   Please enable "osd_enable_op_tracker", and the tracker will start to track new
3 # ceph daemon /var/run/ceph/ceph-osd.0.asok dump_historic_ops
4 op_tracker tracking is not enabled now, so no ops are tracked currently, even those get stuck.   Please enable "osd_enable_op_tracker", and the tracker will start to track new
```

打开 op tracker 后，若集群 iops 很高，osd_num_op_tracker_shard 可以适当调大，因为每个 shard 都有个独立的 mutex 锁；

```
class OpTracker {
...
        struct ShardedTrackingData {
                Mutex ops_in_flight_lock_sharded;
                xlist<TrackedOp *> ops_in_flight_sharded;
                explicit ShardedTrackingData(string lock_name):
                        ops_in_flight_lock_sharded(lock_name.c_str()) {}
        };
        vector<ShardedTrackingData*> sharded_in_flight_list;
        uint32_t num_optracker_shards;
...
};
```

osd_op_threads：对应的 work queue 有 peering_wq（osd peering 请求），recovery_gen_wq（PG recovery 请求）；
osd_disk_threads：对应的 work queue 为 remove_wq（PG remove 请求）；
osd_op_num_shards 和 osd_op_num_threads_per_shard：对应的 thread pool 为 osd_op_tp，work queue 为 op_shardedwq；
处理的请求包括：
OpRequestRef
PGSnapTrim
PGScrub
调大 osd_op_num_shards 可以增大 osd ops 的处理线程数，增大并发性，提升 OSD 性能；

2.2.10 osd client message 配置参数

```
osd_client_message_size_cap = 1048576000       默认值  500*1024L*1024L         // client data allowed in-memory (in bytes)
osd_client_message_cap = 10000                  默认值  100         // num client messages allowed in-memory
```

这个是 osd 端收到 client messages 的 capacity 配置，配置大的话能提升 osd 的处理能力，但会占用较多的系统内存；
配置建议：
服务器内存足够大的时候，适当增大这两个值

2.2.11 osd scrub 配置参数

```
    osd_scrub_begin_hour = 2                    默认值  0
    osd_scrub_end_hour = 6                      默认值  24

    // The time in seconds that scrubbing sleeps between two consecutive scrubs
    osd_scrub_sleep = 2                         默认值  0           // sleep between [deep]scrub ops

    osd_scrub_load_threshold = 5                默认值  0.5

    // chunky scrub 配置的最小/最大 objects 数，以下是默认值
    osd_scrub_chunk_min = 5
    osd_scrub_chunk_max = 25
```

Ceph osd scrub 是保证 ceph 数据一致性的机制，scrub 以 PG 为单位，但每次 scrub 回获取 PG lock，所以它可能会影响 PG 正常的 IO；
Ceph 后来引入了 chunky 的 scrub 模式，每次 scrub 只会选取 PG 的一部分 objects，完成后释放 PG lock，并把下一次的 PG scrub 加入队列；这样能很好的减少 PG scrub 时候占用 PG lock 的时间，避免过多影响 PG 正常的 IO；
同理，引入的 osd_scrub_sleep 参数会让线程在每次 scrub 前释放 PG lock，然后睡眠一段时间，也能很好的减少 scrub 对 PG 正常 IO 的影响；
配置建议：
osd_scrub_begin_hour 和 osd_scrub_end_hour：OSD Scrub 的开始结束时间，根据具体业务指定；
osd_scrub_sleep：osd 在每次执行 scrub 时的睡眠时间；有个 bug 跟这个配置有关，建议关闭；
osd_scrub_load_threshold：osd 开启 scrub 的系统 load 阈值，根据系统的 load average 值配置该参数；
osd_scrub_chunk_min 和 osd_scrub_chunk_max：根据 PG 中 object 的个数配置；针对 RGW 全是小文件的情况，这两个值需要调大；
参考：
http://www.jianshu.com/p/ea2296e1555c
http://tracker.ceph.com/issues/19497

2.2.12 osd thread timeout 配置参数

```
1   osd_op_thread_timeout = 580                     默认值 15
2   osd_op_thread_suicide_timeout = 600             默认值 150
3
4   osd_recovery_thread_timeout = 580               默认值 30
5   osd_recovery_thread_suicide_timeout = 600     默认值 300
```

osd_op_thread_timeout 和 osd_op_thread_suicide_timeout 关联的 work queue 为：
op_shardedwq - 关联的请求为：OpRequestRef，PGSnapTrim，PGScrub
peering_wq - 关联的请求为：osd peering
osd_recovery_thread_timeout 和 osd_recovery_thread_suicide_timeout 关联的 work queue 为：
recovery_wq - 关联的请求为：PG recovery
Ceph 的 work queue 都有个基类 WorkQueue_，定义如下：

```
1     /// Pool of threads that share work submitted to multiple work queues.
2     class ThreadPool : public md_config_obs_t {
3     ...
4           /// Basic interface to a work queue used by the worker threads.
5           struct WorkQueue_ {
6                 string name;
7                 time_t timeout_interval, suicide_interval;
8                 WorkQueue_(string n, time_t ti, time_t sti)
9                       : name(n), timeout_interval(ti), suicide_interval(sti)
10                { }
11  ...
```

这里的 timeout_interval 和 suicide_interval 分别对应上面所述的配置 timeout 和 suicide_timeout；
当 thread 处理 work queue 中的一个请求时，会受到这两个 timeout 时间的限制：
timeout_interval - 到时间后设置 m_unhealthy_workers+1
suicide_interval - 到时间后调用 assert，OSD 进程 crush
对应的处理函数为：

```
1   bool HeartbeatMap::_check(const heartbeat_handle_d *h, const char *who, time_t now)
2   {
3         bool healthy = true;
4         time_t was;
5         was = h->timeout.read();
6         if (was && was < now) {
7               ldout(m_cct, 1) << who << " '" << h->name << "'"
8                                     << " had timed out after " << h->grace << dendl;
9               healthy = false;
10        }
11        was = h->suicide_timeout.read();
12        if (was && was < now) {
13              ldout(m_cct, 1) << who << " '" << h->name << "'"
14                                    << " had suicide timed out after " << h->suicide_grace << dendl;
15              assert(0 == "hit suicide timeout");
16        }
17        return healthy;
18  }
```

当前仅有 RGW 添加了 worker 的 perfcounter，所以也只有 RGW 可以通过 perf dump 查看 total/unhealthy 的 worker 信息：

```
1 [root@ yangguanjun]# ceph daemon /var/run/ceph/ceph-client.rgw.rgwdaemon.asok perf dump | grep worker
2             "total_workers": 32,
3             "unhealthy_workers": 0
```

对应的配置项为：

```
1   OPTION(rgw_num_async_rados_threads, OPT_INT, 32) // num of threads to use for async rados operations
2   ```
3
4   **配置建议：**
5
6   - `*_thread_timeout`：这个值配置越小越能及时发现处理慢的请求，所以不建议配置很大；特别是针对速度快的设备，建议调小该值；
7   - `*_thread_suicide_timeout`：这个值配置小了会导致超时后的 OSD crush，所以建议调大；特别是在对应的 throttle 调大后，更应该调大该值；
```

```
8
9   ### 13，fielstore op thread 配置参数
10  ```sh
11  filestore_op_threads = 10                          默认值 2
12  filestore_op_thread_timeout = 580                  默认值 60
13  filestore_op_thread_suicide_timeout = 600     默认值 180
```

filestore_op_threads：对应的 thread pool 为 op_tp，对应的 work queue 为 op_wq；filestore 的所有请求都经过 op_wq 处理；
增大该参数能提升 filestore 的处理能力，提升 filestore 的性能；配合 filestore 的 throttle 一起调整；
filestore_op_thread_timeout 和 filestore_op_thread_suicide_timeout 关联的 work queue 为：op_wq
配置的含义与上一节中的 thread_timeout/thread_suicide_timeout 保持一致；

2.2.13 filestore merge/split 配置参数

```
1   filestore_merge_threshold = -1          默认值 10
2   filestore_split_multiple = 16000        默认值 2
```

这两个参数是管理 filestore 的目录分裂/合并的，filestore 的每个目录允许的最大文件数为：
filestore_split_multiple * abs(filestore_merge_threshold) * 16
在 RGW 的小文件应用场景，会很容易达到默认配置的文件数（320），若在写的过程中触发了 filestore 的分裂，则会非常影响 filestore 的性能；
每次 filestore 的目录分裂，会依据如下规则分裂为多层目录，最底层 16 个子目录：
例如 PG 31.4C0, hash 结尾是 4C0，若该目录分裂，会分裂为 DIR_0/DIR_C/DIR_4/{DIR_0, DIR_F}；
原始目录下的 object 会根据规则放到不同的子目录里，object 的名称格式为: *__head_xxxxX4C0_*，分裂时候 X 是几，就放进子目录 DIR_X 里。比如 object：*__head_xxxxA4C0_*，就放进子目录 DIR_0/DIR_C/DIR_4/DIR_A 里；
解决办法：
增大 merge/split 配置参数的值，使单个目录容纳更多的文件；
filestore_merge_threshold 配置为负数；这样会提前触发目录的预分裂，避免目录在某一时间段的集中分裂，详细机制没有调研；
创建 pool 时指定 expected-num-objects；这样会依据目录分裂规则，在创建 pool 的时候就创建分裂的子目录，避免了目录分裂对 filestore 性能的影响；
参考：
http://docs.ceph.com/docs/master/rados/configuration/filestore-config-ref/
http://docs.ceph.com/docs/jewel/rados/operations/pools/#create-a-pool
http://blog.csdn.net/for_tech/article/details/51251936
http://ivanjobs.github.io/page3/

2.2.14 filestore fd cache 配置参数

```
1  filestore_fd_cache_shards =    32        默认值 16        // FD number of shards
2  filestore_fd_cache_size = 32768          默认值 128    // FD lru size
```

filestore 的 fd cache 是加速访问 filestore 里的 file 的，在非一次性写入的应用场景，增大配置可以很明显的提升 filestore 的性能；

2.2.15 filestore sync 配置参数

```
1 filestore_wbthrottle_enable = false       默认值 true              SSD 的时候建议关闭
2 filestore_min_sync_interval = 5           默认值 0.01 s       最小同步间隔秒数，sync fs 的数据到 disk，FileStore::sync_entry()
3 filestore_max_sync_interval = 10          默认值 5 s          最大同步间隔秒数，sync fs 的数据到 disk，FileStore::sync_entry()
4 filestore_commit_timeout = 3000           默认值 600 s          FileStore::sync_entry() 里 new SyncEntryTimeout(m_filestore_commit_timeout)
```

filestore_wbthrottle_enable 的配置是关于 filestore writeback throttle 的，即我们说的 filestore 处理 workqueue op_wq 的数据量阈值；默认值是 true，开启后 XFS 相关的配置参数有：

```
1   OPTION(filestore_wbthrottle_xfs_bytes_start_flusher, OPT_U64, 41943040)
2   OPTION(filestore_wbthrottle_xfs_bytes_hard_limit, OPT_U64, 419430400)
3   OPTION(filestore_wbthrottle_xfs_ios_start_flusher, OPT_U64, 500)
4   OPTION(filestore_wbthrottle_xfs_ios_hard_limit, OPT_U64, 5000)
5   OPTION(filestore_wbthrottle_xfs_inodes_start_flusher, OPT_U64, 500)
6   OPTION(filestore_wbthrottle_xfs_inodes_hard_limit, OPT_U64, 5000)
```

若使用普通 HDD，可以保持其为 true；针对 SSD，建议将其关闭，不开启 writeback throttle；
filestore_min_sync_interval 和 filestore_max_sync_interval 是配置 filestore flush outstanding IO 到 disk 的时间间隔的；增大配置可以让系统做尽可能多的 IO merge，减少 filestore 写磁盘的压力，但也会增大 page cache 占用内存的开销，增大数据丢失的可能性；
filestore_commit_timeout 是配置 filestore sync entry 到 disk 的超时时间，在 filestore 压力很大时，调大这个值能尽量避免 IO 超时导致 OSD crush；

2.2.16 filestore throttle 配置参数

```
filestore_expected_throughput_bytes =    536870912          默认值  200MB       /// Expected filestore throughput in B/s
filestore_expected_throughput_ops = 2500                     默认值  200         /// Expected filestore throughput in ops/s
filestore_queue_max_bytes= 1048576000                          默认值  100MB
filestore_queue_max_ops = 5000                                 默认值  50

/// Use above to inject delays intended to keep the op queue between low and high
filestore_queue_low_threshhold = 0.6                         默认值  0.3
filestore_queue_high_threshhold = 0.9                        默认值  0.9

filestore_queue_high_delay_multiple = 2                        默认值  0       /// Filestore high delay multiple.    Defaults to 0 (disabled)
filestore_queue_max_delay_multiple = 10                      默认值  0       /// Filestore max delay multiple.    Defaults to 0 (disabled)
```

在 jewel 版本里，引入了 dynamic throttle，来平滑普通 throttle 带来的长尾效应问题；

一般在使用普通磁盘时，之前的 throttle 机制即可很好的工作，所以这里默认 filestore_queue_high_delay_multiple 和 filestore_queue_max_delay_multiple 都为 0；

针对高速磁盘，需要在部署之前，通过小工具 ceph_smalliobenchfs 来测试下，获取合适的配置参数；

```
    BackoffThrottle 的介绍如下：
    /**
    * BackoffThrottle
    *
    * Creates a throttle which gradually induces delays when get() is called
    * based on params low_threshhold, high_threshhold, expected_throughput,
    * high_multiple, and max_multiple.
    *
    * In [0, low_threshhold), we want no delay.
    *
    * In [low_threshhold, high_threshhold), delays should be injected based
    * on a line from 0 at low_threshhold to
    * high_multiple * (1/expected_throughput) at high_threshhold.
    *
    * In [high_threshhold, 1), we want delays injected based on a line from
    * (high_multiple * (1/expected_throughput)) at high_threshhold to
    * (high_multiple * (1/expected_throughput)) +
    * (max_multiple * (1/expected_throughput)) at 1.
    *
    * Let the current throttle ratio (current/max) be r, low_threshhold be l,
    * high_threshhold be h, high_delay (high_multiple / expected_throughput) be e,
    * and max_delay (max_muliple / expected_throughput) be m.
    *
    * delay = 0, r \in [0, l)
    * delay = (r - l) * (e / (h - l)), r \in [l, h)
    * delay = h + (r - h)((m - e)/(1 - h))
    */
```

参考：

http://docs.ceph.com/docs/jewel/dev/osd_internals/osd_throttles/

http://blog.wjin.org/posts/ceph-dynamic-throttle.html

https://github.com/ceph/ceph/blob/master/src/doc/dynamic-throttle.txt

Ceph BackoffThrottle 分析

2.2.17 filestore finisher threads 配置参数

```
filestore_ondisk_finisher_threads = 2     默认值  1
filestore_apply_finisher_threads = 2     默认值  1
```

这两个参数定义 filestore commit/apply 的 finisher 处理线程数，默认都为 1，任何 IO commit/apply 完成后，都需要经过对应的 ondisk/apply finisher thread 处理；

在使用普通 HDD 时，磁盘性能是瓶颈，单个 finisher thread 就能处理好；

但在使用高速磁盘的时候，IO 完成比较快，单个 finisher thread 不能处理这么多的 IO commit/apply reply，它会成为瓶颈；所以在 jewel 版本里引入了 finisher thread pool 的配置，这里一般配置为 2 即可；

2.2.18 journal 配置参数

```
journal_max_write_bytes=1048576000              默认值  10M
journal_max_write_entries=5000                   默认值  100

journal_throttle_high_multiple = 2              默认值  0       /// Multiple over expected at high_threshhold. Defaults to 0 (disabled).
journal_throttle_max_multiple = 10               默认值  0       /// Multiple over expected at max.    Defaults to 0 (disabled).

/// Target range for journal fullness
OPTION(journal_throttle_low_threshhold, OPT_DOUBLE, 0.6)
OPTION(journal_throttle_high_threshhold, OPT_DOUBLE, 0.9)
```

journal_max_write_bytes 和 journal_max_write_entries 是 journal 一次 write 的数据量和 entries 限制；
针对 SSD 分区做 journal 的情况，这两个值要增大，这样能增大 journal 的吞吐量；
journal_throttle_high_multiple 和 journal_throttle_max_multiple 是 JournalThrottle 的配置参数，JournalThrottle 是 BackoffThrottle 的封装类，所以 JournalThrottle 与我们在 filestore throttle 介绍的 dynamic throttle 工作原理一样；

```
int FileJournal::set_throttle_params()
{
        stringstream ss;
        bool valid = throttle.set_params(
                                g_conf->journal_throttle_low_threshhold,
                                g_conf->journal_throttle_high_threshhold,
                                g_conf->filestore_expected_throughput_bytes,
                                g_conf->journal_throttle_high_multiple,
                                g_conf->journal_throttle_max_multiple,
                                header.max_size - get_top(),
                                &ss);
...
}
```

从上述代码中看出相关的配置参数有：
journal_throttle_low_threshhold
journal_throttle_high_threshhold
filestore_expected_throughput_bytes

2.2.19 rbd cache 配置参数

```
[client]
rbd_cache_size = 134217728                      默认值  32M // cache size in bytes
rbd_cache_max_dirty = 100663296                 默认值  24M // dirty limit in bytes - set to 0 for write-through caching
rbd_cache_target_dirty = 67108864               默认值  16M // target dirty limit in bytes
rbd_cache_writethrough_until_flush = true       默认值  true       // whether to make writeback caching writethrough until flush is called, to be sure the user of librbd will
send flushs so that writeback is safe
rbd_cache_max_dirty_age = 5                     默认值  1.0        // seconds in cache before writeback starts
```

rbd_cache_size：client 端每个 rbd image 的 cache size，不需要太大，可以调整为 64M，不然会比较占 client 端内存；
参照默认值，根据 rbd_cache_size 的大小调整 rbd_cache_max_dirty 和 rbd_cache_target_dirty；
rbd_cache_max_dirty：在 writeback 模式下 cache 的最大 bytes 数，默认是 24MB；当该值为 0 时，表示使用 writethrough 模式；
rbd_cache_target_dirty：在 writeback 模式下 cache 向 ceph 集群写入的 bytes 阀值，默认 16MB；注意该值一定要小于 rbd_cache_max_dirty 值
rbd_cache_writethrough_until_flush：在内核触发 flush cache 到 ceph 集群前 rbd cache 一直是 writethrough 模式，直到 flush 后 rbd cache 变成 writeback 模式；
rbd_cache_max_dirty_age：标记 OSDC 端 ObjectCacher 中 entry 在 cache 中的最长时间；
可以尝试一下是否有效果。

2.2.20 PG number

PG 和 PGP 数量一定要根据 OSD 的数量进行调整，计算公式如下，但是最后算出的结果一定要接近或者等于一个 2 的指数。
Total PGs = (Total_number_of_OSD * 100) / max_replication_count
例：
有 100 个 osd，2 副本，5 个 pool
Total PGs =100*100/2=5000
每个 pool 的 PG=5000/5=1000，那么创建 pool 的时候就指定 pg 为 1024
ceph osd pool create pool_name 1024
当前块存储系统保持现有配置即可。

2.2.21 修改 crushmap

Crush map 可以设置不同的 osd 对应到不同的 pool，也可以修改每个 osd 的 weight
配置可参考：http://linuxnote.blog.51cto.com/9876511/1790758
当前块存储系统保持现有配置即可。

2.2.22 bluestore 压缩数据优化参数

参考链接: https://lovethegirl.github.io/2020/02/11/compression/

2.2.23 其他

ceph osd perf
通过 osd perf 可以提供磁盘 latency 的状况，如果延时过长，应该剔除 osd。

三、历史经验

bluestore_csum_type = crc32c_16
bluestore_min_alloc_size = 65536
bluestore_cache_size_ssd = 536870912
bluestore_cache_size_hdd = 268435456
osd_min_pg_log_entries = 100
osd_max_pg_log_entries = 100
osd_pg_log_dups_tracked = 100
bluestore_rocksdb_options=
compression=kNoCompression,max_write_buffer_number=3,min_write_buffer_number_to_merge=1,recycle_log_file_num=4,write_buffer_size=134217728,writable_file_max_buffer_size=0,compaction_readahead_size=131072,max_background_compactions=1,max_background_flushes=1

四、参考资料

1. https://docs.ceph.com/en/reef/rados/
2. https://blog.csdn.net/changtao381/article/details/49907115
3. https://blog.csdn.net/don_chiang709/category_8958706.html