[**理解 OpenStack + Ceph （8）: 基本的 Ceph 性能测试工具和方法**](http://www.cnblogs.com/sammyliu/p/5557666.html)

本系列文章会深入研究 Ceph 以及 Ceph 和 OpenStack 的集成：

（1）[安装和部署](http://www.cnblogs.com/sammyliu/p/4804037.html%20)

（2）[Ceph RBD 接口和工具](http://www.cnblogs.com/sammyliu/p/4838139.html%20)

（3）[Ceph 物理和逻辑结构](http://www.cnblogs.com/sammyliu/p/4836014.html%20)

（4）[Ceph 的基础数据结构](http://www.cnblogs.com/sammyliu/p/4843812.html%20)

（5）[Ceph 与 OpenStack 集成的实现](http://www.cnblogs.com/sammyliu/p/4838138.html)

（6）[QEMU-KVM 和 Ceph RBD 的 缓存机制总结](http://www.cnblogs.com/sammyliu/p/5066895.html)

（7）[Ceph 的基本操作和常见故障排除方法](http://www.cnblogs.com/sammyliu/p/5555218.html%20)

（8）[基本的性能测试工具和方法](http://www.cnblogs.com/sammyliu/p/5557666.html%20)

继续学以致用，学习下基本的Ceph性能测试工具和方法。

**0. 测试环境**

同 [Ceph 的基本操作和常见故障排除方法](http://www.cnblogs.com/sammyliu/p/5555218.html%20) 一文中的测试环境。

**1. 测试准备**

**1.1 磁盘读写性能**

**1.1.1 单个 OSD 磁盘写性能，大概 165MB/s。**

root@ceph1:~# echo 3 > /proc/sys/vm/drop\_caches

root@ceph1:~# dd if=/dev/zero of=/var/lib/ceph/osd/ceph-0/deleteme bs=1G count=1 oflag=direct

测试发现，其结果变化非常大，有时候上 75，有时是150.

**1.1.2 两个OSD同时写性能，大概 18 MB/s。怎么差距那么大呢？几乎是单个磁盘的 1/10 了。**

root@ceph1:~# for i in `mount | grep osd | awk '{print $3}'`; do (dd if=/dev/zero of=$i/deleteme bs=1G count=1 oflag=direct &) ; done

**1.1.4 单个 OSD 磁盘读性能，大概 460 MB/s。**

root@ceph1:~# dd if=/var/lib/ceph/osd/ceph-0/deleteme of=/dev/null bs=2G count=1 iflag=direct

**1.1.5 两个 OSD 同时读性能，大概 130 MB/s。**

for i in `mount | grep osd | awk '{print $3}'`; do (dd if=$i/deleteme of=/dev/null bs=1G count=1 iflag=direct &); done

**1.2 网络性能**

在 ceph1上运行 iperf -s -p 6900，在 ceph2 上运行 iperf -c ceph1 -p 6900，反复多次，两节点之间的带宽大约在 1 Gbits/sec = 128 MB/s。

[复制代码](javascript:void(0);)

root@ceph2:~# iperf -c ceph1 -p 6900

------------------------------------------------------------

Client connecting to ceph1, TCP port 6900

TCP window size: 85.0 KByte (default)

------------------------------------------------------------

[ 3] local 192.168.56.103 port 41773 connected with 192.168.56.102 port 6900

[ ID] Interval Transfer Bandwidth

[ 3] 0.0-10.0 sec 1.25 GBytes 1.08 Gbits/sec

[复制代码](javascript:void(0);)

**2. Ceph 性能测试**

**2.1 RADOS 性能测试：使用 Ceph 自带的 rados bench 工具**

该工具的语法为：rados bench -p <pool\_name> <seconds> <write|seq|rand> -b <block size> -t --no-cleanup

* pool\_name：测试所针对的存储池
* seconds：测试所持续的秒数
* <write|seq|rand>：操作模式，write：写，seq：顺序读；rand：随机读
* -b：block size，即块大小，默认为 4M
* -t：读/写并行数，默认为 16
* --no-cleanup 表示测试完成后不删除测试用数据。在做读测试之前，需要使用该参数来运行一遍写测试来产生测试数据，在全部测试结束后可以运行 rados -p <pool\_name> cleanup 来清理所有测试数据。

写：

[复制代码](javascript:void(0);)

root@ceph1:~# rados bench -p rbd 10 write --no-cleanup

Maintaining 16 concurrent writes of 4194304 bytes for up to 10 seconds or 0 objects

Object prefix: benchmark\_data\_ceph1\_12884

sec Cur ops started finished avg MB/s cur MB/s last lat avg lat

0 0 0 0 0 0 - 0

1 16 16 0 0 0 - 0

...

12 15 75 60 19.9671 4 3.05943 2.46556

Total time run: 12.135344

Total writes made: 75

Write size: 4194304

Bandwidth (MB/sec): 24.721

Stddev Bandwidth: 13.5647

Max bandwidth (MB/sec): 36

Min bandwidth (MB/sec): 0

Average Latency: 2.57614

Stddev Latency: 0.781915

Max latency: 4.50816

Min latency: 1.04075

[复制代码](javascript:void(0);)

顺序读：

[复制代码](javascript:void(0);)

root@ceph1:~# rados bench -p rbd 10 seq

sec Cur ops started finished avg MB/s cur MB/s last lat avg lat

0 16 16 0 0 0 - 0

Total time run: 0.601027

Total reads made: 75

Read size: 4194304

Bandwidth (MB/sec): 499.146

Average Latency: 0.123632

Max latency: 0.209325

Min latency: 0.030446

[复制代码](javascript:void(0);)

随机读：

[复制代码](javascript:void(0);)

root@ceph1:~# rados bench -p rbd 10 rand

sec Cur ops started finished avg MB/s cur MB/s last lat avg lat

0 3 3 0 0 0 - 0

1 16 138 122 477.298 488 0.01702 0.116519

...

10 16 1242 1226 488.681 448 0.108589 0.129214

Total time run: 10.092985

Total reads made: 1242

Read size: 4194304

Bandwidth (MB/sec): 492.223

Average Latency: 0.129631

Max latency: 0.297213

Min latency: 0.007133

[复制代码](javascript:void(0);)

**2.2 RADOS 性能测试：使用 rados load-gen 工具**

该工具的语法为：

[复制代码](javascript:void(0);)

# rados -p rbd load-gen

--num-objects     初始生成测试用的对象数，默认 200  
--min-object-size 测试对象的最小大小，默认 1KB，单位byte   
--max-object-size 测试对象的最大大小，默认 5GB，单位byte

--min-op-len      压测IO的最小大小，默认 1KB，单位byte  
--max-op-len      压测IO的最大大小，默认 2MB，单位byte  
--max-ops         一次提交的最大IO数，相当于iodepth  
--target-throughput 一次提交IO的历史累计吞吐量上限，默认 5MB/s，单位B/s  
--max-backlog     一次提交IO的吞吐量上限，默认10MB/s，单位B/s  
--read-percent    读写混合中读的比例，默认80，范围[0, 100]

--run-length      运行的时间，默认60s，单位秒

[复制代码](javascript:void(0);)

在 ceph1上运行 rados -p pool100 load-gen --read-percent 0 --min-object-size 1073741824 --max-object-size 1073741824 --max-ops 1 --read-percent 0 --min-op-len 4194304 --max-op-len 4194304 --target-throughput 1073741824 --max\_backlog 1073741824 的结果为：

[复制代码](javascript:void(0);)

WRITE : oid=obj-y0UPAZyRQNhnabq off=929764660 len=4194304  
op 19 completed, throughput=16MB/sec  
WRITE : oid=obj-nPcOZAc4ebBcnyN off=143211384 len=4194304  
op 20 completed, throughput=20MB/sec  
WRITE : oid=obj-sWGUAzzASPjCcwF off=343875215 len=4194304  
op 21 completed, throughput=24MB/sec  
WRITE : oid=obj-79r25fxxSMgVm11 off=383617425 len=4194304  
op 22 completed, throughput=28MB/sec

[复制代码](javascript:void(0);)

该命令的含义是：在 1G 的对象上，以 iodepth = 1 顺序写入 block size 为 4M 的总量为 1G 的数据。其平均结果大概在 24MB/s，基本和 rados bench 的结果相当。

在 client 上，同样的配置，顺序写的BW大概在 20MB/s，顺序读的 BW 大概在 100 MB/s。

可见，与 rados bench 相比，rados load-gen 的特点是可以产生混合类型的测试负载，而 rados bench 只能产生一种类型的负载。但是 load-gen 只能输出吞吐量，只合适做类似于 4M 这样的大block size 数据测试，输出还不包括延迟。

**2.3 使用 rbd bench-write 进行块设备写性能测试**

**2.3.1 客户端准备**

   在执行如下命令来准备 Ceph 客户端：

[复制代码](javascript:void(0);)

root@client:/var# rbd create bd2 --size 1024

root@client:/var# rbd info --image bd2

rbd image 'bd2':

size 1024 MB in 256 objects

order 22 (4096 kB objects)

block\_name\_prefix: rb.0.3841.74b0dc51

format: 1

root@client:/var# rbd map bd2

root@client:/var# rbd showmapped

id pool image snap device

1 pool1 bd1 - /dev/rbd1

2 rbd bd2 - /dev/rbd2

root@client:/var# mkfs.xfs /dev/rbd2

log stripe unit (4194304 bytes) is too large (maximum is 256KiB)

log stripe unit adjusted to 32KiB

meta-data=/dev/rbd2 isize=256 agcount=9, agsize=31744 blks

= sectsz=512 attr=2, projid32bit=0

data = bsize=4096 blocks=262144, imaxpct=25

= sunit=1024 swidth=1024 blks

naming =version 2 bsize=4096 ascii-ci=0

log =internal log bsize=4096 blocks=2560, version=2

= sectsz=512 sunit=8 blks, lazy-count=1

realtime =none extsz=4096 blocks=0, rtextents=0

root@client:/var# mkdir -p /mnt/ceph-bd2

root@client:/var# mount /dev/rbd2 /mnt/ceph-bd2/

root@client:/var# df -h /mnt/ceph-bd2/

Filesystem Size Used Avail Use% Mounted on

/dev/rbd2 1014M 33M 982M 4% /mnt/ceph-bd2

[复制代码](javascript:void(0);)

**2.3.2 测试**

rbd bench-write 的语法为：rbd bench-write <RBD image name>，可以带如下参数：

* --io-size：单位 byte，默认 4096 bytes = 4K
* --io-threads：线程数，默认 16
* --io-total：总写入字节，单位为字节，默认 1024M
* --io-pattern <seq|rand>：写模式，默认为 seq 即顺序写

 分别在集群 OSD 节点上和客户端上做测试：

（1）在 OSD 节点上做测试

[复制代码](javascript:void(0);)

root@ceph1:~# rbd bench-write bd2 --io-total 171997300

bench-write io\_size 4096 io\_threads 16 bytes 171997300 pattern seq

SEC OPS OPS/SEC BYTES/SEC

1 280 273.19 2237969.65

2 574 286.84 2349818.65

...

71 20456 288.00 2358395.28

72 20763 288.29 2360852.64

elapsed: 72 ops: 21011 ops/sec: 288.75 bytes/sec: 2363740.27

[复制代码](javascript:void(0);)

此时，块大小为 4k，IOPS 为 289，BW 为 2.36 MB/s （怎么 BW 是 block\_size \* IOPS 的两倍呢？）。

（2）在客户端上做测试

[复制代码](javascript:void(0);)

root@client:/home/s1# rbd bench-write pool.host/image.ph2 --io-total 1719973000 --io-size 4096000

bench-write io\_size 4096000 io\_threads 16 bytes 1719973000 pattern seq

SEC OPS OPS/SEC BYTES/SEC

1 5 3.41 27937685.86

2 19 9.04 68193147.96

3 28 8.34 62237889.75

5 36 6.29 46538807.31

...

39 232 5.86 40792216.64

40 235 5.85 40666942.19

elapsed: 41 ops: 253 ops/sec: 6.06 bytes/sec: 41238190.87

[复制代码](javascript:void(0);)

此时 block size 为 4M，IOPS 为 6， BW 为 41.24 MB/s。

[复制代码](javascript:void(0);)

root@client:/home/s1# rbd bench-write pool.host/image.ph2 --io-total 1719973000

bench-write io\_size 4096 io\_threads 16 bytes 1719973000 pattern seq

SEC OPS OPS/SEC BYTES/SEC

1 331 329.52 2585220.17

2 660 329.57 2521925.67

3 1004 333.17 2426190.82

4 1331 332.26 2392607.58

5 1646 328.68 2322829.13

6 1986 330.88 2316098.66

[复制代码](javascript:void(0);)

此时 block size 为 4K，IOPS 为 330 左右， BW 为 24 MB/s 左右。

备注：从 [rbd bench-write vs dd performance confusion](http://www.spinics.net/lists/ceph-users/msg24647.html) 中看起来，rados bench-write 似乎有bug。我所使用的Ceph 是0.80.11 版本，可能补丁还没有合进来。

**2.4 使用  fio +rbd ioengine**

**2.4.1 环境准备**

运行 apt-get install fio 来安装 fio 工具。创建 fio 配置文件：

[复制代码](javascript:void(0);)

root@client:/home/s1# cat write.fio

[write-4M]

description="write test with block size of 4M"

ioengine=rbd

clientname=admin

pool=rbd

rbdname=bd2

iodepth=32

runtime=120

rw=write #write 表示顺序写，randwrite 表示随机写，read 表示顺序读，randread 表示随机读

bs=4M

[复制代码](javascript:void(0);)

运行 fio 命令，但是出错：

[复制代码](javascript:void(0);)

root@client:/home/s1# fio write.fio

fio: engine rbd not loadable

fio: failed to load engine rbd

Bad option <clientname=admin>

Bad option <pool=rbd>

Bad option <rbdname=bd2>

fio: job write-4M dropped

fio: file:ioengines.c:99, func=dlopen, error=rbd: cannot open shared object file: No such file or directory

[复制代码](javascript:void(0);)

其原因是因为没有安装 fio librbd IO 引擎，因此当前 fio 无法支持 rbd ioengine：

[复制代码](javascript:void(0);)

root@client:/home/s1# fio --enghelp

Available IO engines:

cpuio

mmap

sync

psync

vsync

pvsync

null

net

netsplice

libaio

rdma

posixaio

falloc

e4defrag

splice

sg

binject

[复制代码](javascript:void(0);)

在运行 apt-get install librbd-dev 命令安装 librbd 后，fio 还是报同样的错误。参考网上资料，下载 fio 代码重新编译 fio：

[复制代码](javascript:void(0);)

$ git clone git://git.kernel.dk/fio.git

$ cd fio

$ ./configure

[...]

Rados Block Device engine yes

[...]

$ make

[复制代码](javascript:void(0);)

此时 fio 的 ioengine 列表中也有 rbd 了。fio 使用 rbd IO 引擎后，它会读取 ceph.conf 中的配置去连接 Ceph 集群。

下面是 fio 命令和结果：

[复制代码](javascript:void(0);)

root@client:/home/s1/fio# ./fio ../write.fio

write-4M: (g=0): rw=write, bs=4M-4M/4M-4M/4M-4M, ioengine=rbd, iodepth=32

fio-2.11-12-g82e6

Starting 1 process

rbd engine: RBD version: 0.1.8

Jobs: 1 (f=1): [W(1)] [100.0% done] [0KB/128.0MB/0KB /s] [0/32/0 iops] [eta 00m:00s]

write-4M: (groupid=0, jobs=1): err= 0: pid=19190: Sat Jun 4 22:30:00 2016

Description : ["write test with block size of 4M"]

write: io=1024.0MB, bw=17397KB/s, iops=4, runt= 60275msec

slat (usec): min=129, max=54100, avg=1489.10, stdev=4907.83

clat (msec): min=969, max=15690, avg=7399.86, stdev=1328.55

lat (msec): min=969, max=15696, avg=7401.35, stdev=1328.67

clat percentiles (msec):

| 1.00th=[ 971], 5.00th=[ 6325], 10.00th=[ 6325], 20.00th=[ 6521],

| 30.00th=[ 6718], 40.00th=[ 7439], 50.00th=[ 7439], 60.00th=[ 7635],

| 70.00th=[ 7832], 80.00th=[ 8291], 90.00th=[ 8356], 95.00th=[ 8356],

| 99.00th=[14615], 99.50th=[15664], 99.90th=[15664], 99.95th=[15664],

| 99.99th=[15664]

bw (KB /s): min=245760, max=262669, per=100.00%, avg=259334.50, stdev=6250.72

lat (msec) : 1000=1.17%, >=2000=98.83%

cpu : usr=0.24%, sys=0.03%, ctx=50, majf=0, minf=8

IO depths : 1=2.3%, 2=5.5%, 4=12.5%, 8=25.0%, 16=50.4%, 32=4.3%, >=64=0.0%

submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%

complete : 0=0.0%, 4=97.0%, 8=0.0%, 16=0.0%, 32=3.0%, 64=0.0%, >=64=0.0%

issued : total=r=0/w=256/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0

latency : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):

WRITE: io=1024.0MB, aggrb=17396KB/s, minb=17396KB/s, maxb=17396KB/s, mint=60275msec, maxt=60275msec

Disk stats (read/write):

sda: ios=0/162, merge=0/123, ticks=0/19472, in\_queue=19472, util=6.18%

[复制代码](javascript:void(0);)

如果 iodepth = 1 的话，结果是：

[复制代码](javascript:void(0);)

root@client:/home/s1# fio/fio write.fio.dep1

write-4M: (g=0): rw=write, bs=4M-4M/4M-4M/4M-4M, ioengine=rbd, iodepth=1

fio-2.11-12-g82e6

Starting 1 process

rbd engine: RBD version: 0.1.8

Jobs: 1 (f=1): [W(1)] [100.0% done] [0KB/8192KB/0KB /s] [0/2/0 iops] [eta 00m:00s]

write-4M: (groupid=0, jobs=1): err= 0: pid=19250: Sat Jun 4 22:33:11 2016

Description : ["write test with block size of 4M"]

write: io=1024.0MB, bw=20640KB/s, iops=5, runt= 50802msec

[复制代码](javascript:void(0);)

**2.5 使用 fio + libaio 进行测试**

libaio 是 Linux native asynchronous I/O。

几种测试模式：

* 随机写：fio/fio -filename=/mnt/ceph-rbd2 -direct=1 -iodepth 1 -thread -rw=randwrite -ioengine=libaio -bs=4M -size=1G -numjobs=1 -runtime=120 -group\_reporting -name=read-libaio

 这些参数的含义是：

* + filename：表示待测试的设备名称。
  + iodepth： libaio 会用这个 iodepth 值来调用 io\_setup 准备个可以一次提交 iodepth 个 IO 的上下文，同时申请个io请求队列用于保持IO。
  + iodepth\_batch：在压测进行的时候，系统会生成特定的IO请求，往io请求队列里面扔，当队列里面的IO个数达到 iodepth\_batch 值的时候，
  + iodepth\_batch\_complete 和 iodepth\_low： 调用 io\_submit 批次提交请求，然后开始调用 io\_getevents 开始收割已经完成的IO。 每次收割多少呢？由于收割的时候，超时时间设置为0，所以有多少已完成就算多少，最多可以收割 iodepth\_batch\_complete 值个。随着收割，IO队列里面的IO数就少了，那么需要补充新的IO。 什么时候补充呢？当IO数目降到 iodepth\_low 值的时候，就重新填充，保证 OS 可以看到至少 iodepth\_low 数目的io在电梯口排队着。
* [复制代码](javascript:void(0);)
* root@client:/home/s1# fio/fio -filename=/mnt/ceph-rbd2 -direct=1 -iodepth 1 -thread -rw=randwrite -ioengine=libaio -bs=4M -size=1G -numjobs=1 -runtime=120 -group\_reporting -name=read-libaio read-libaio: (g=0): rw=randwrite, bs=4M-4M/4M-4M/4M-4M, ioengine=libaio, iodepth=1
* fio-2.11-12-g82e6
* Starting 1 thread
* Jobs: 1 (f=1): [w(1)] [100.0% done] [0KB/94302KB/0KB /s] [0/23/0 iops] [eta 00m:00s]
* read-libaio: (groupid=0, jobs=1): err= 0: pid=20256: Sun Jun 5 10:00:55 2016
* write: io=1024.0MB, bw=102510KB/s, iops=25, runt= 10229msec
* slat (usec): min=342, max=5202, avg=1768.90, stdev=1176.00
* clat (usec): min=332, max=165391, avg=38165.11, stdev=27987.64
* lat (msec): min=3, max=167, avg=39.94, stdev=28.00
* clat percentiles (msec):
* | 1.00th=[ 8], 5.00th=[ 18], 10.00th=[ 19], 20.00th=[ 20],
* | 30.00th=[ 22], 40.00th=[ 25], 50.00th=[ 29], 60.00th=[ 31],
* | 70.00th=[ 36], 80.00th=[ 47], 90.00th=[ 83], 95.00th=[ 105],
* | 99.00th=[ 123], 99.50th=[ 131], 99.90th=[ 165], 99.95th=[ 165],
* | 99.99th=[ 165]
* bw (KB /s): min=32702, max=172032, per=97.55%, avg=99999.10, stdev=36075.23
* lat (usec) : 500=0.39%
* lat (msec) : 4=0.39%, 10=0.39%, 20=21.48%, 50=57.81%, 100=14.45%
* lat (msec) : 250=5.08%
* cpu : usr=0.62%, sys=3.65%, ctx=316, majf=0, minf=9
* IO depths : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
* submit : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
* complete : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
* issued : total=r=0/w=256/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0
* latency : target=0, window=0, percentile=100.00%, depth=1
* Run status group 0 (all jobs):
* WRITE: io=1024.0MB, aggrb=102510KB/s, minb=102510KB/s, maxb=102510KB/s, mint=10229msec, maxt=10229msec
* Disk stats (read/write):

sda: ios=0/1927, merge=0/1, ticks=0/30276, in\_queue=30420, util=98.71%

[复制代码](javascript:void(0);)

* 随机读：fio/fio -filename=/mnt/ceph-rbd2 -direct=1 -iodepth 1 -thread -rw=randread -ioengine=libaio -bs=4M -size=1G -numjobs=1 -runtime=120 -group\_reporting -name=read-libaio
* 顺序写：fio/fio -filename=/mnt/ceph-rbd2 -direct=1 -iodepth 1 -thread -rw=write -ioengine=libaio -bs=4M -size=1G -numjobs=1 -runtime=120 -group\_reporting -name=read-libaio
* 随机写：fio/fio -filename=/mnt/ceph-rbd2 -direct=1 -iodepth 1 -thread -rw=randwrite -ioengine=libaio -bs=4M -size=1G -numjobs=1 -runtime=120 -group\_reporting -name=read-libaio

**3. 总结**

**3.1 测试工具小结**

|  |  |  |  |
| --- | --- | --- | --- |
| **工具** | **用途** | **语法** | **说明** |
| dd | 磁盘读写性能测试 | dd if=/dev/zero of=/root/testfile bs=1G count=1 oflag=direct/dsync/sync | <https://www.thomas-krenn.com/en/wiki/Linux_I/O_Performance_Tests_using_dd> |
| iperf | 网络带宽性能测试 |  | <https://iperf.fr/> |
| rados bench | RADOS 性能测试工具 | rados bench -p <pool\_name> <seconds> <write|seq|rand> -b <block size> -t --no-cleanup | * Ceph 自带的 RADOS 性能测试工具 |
| rados load-gen | RADOS 性能测试工具 | # rados -p rbd load-gen  --num-objects #产生的对象数目  --min-object-size #最小对象大小  --max-object-size #最大对象大小  --max-ops #最大操作数目  --min-op-len #最小操作长度  --max-op-len #最大操作长度  --read-percent #读操作的百分比  --target-throughput #目标吞吐量，单位 MB  --run-length #运行时长，单位秒 | * Ceph 自带的 rados 性能测试工具 * 可在集群内产生指定类型的负载 * 比 rados bench 功能丰富，能指定更多的参数 |
| rbd bench-write | ceph 自带的 rbd 性能测试工具 | rbd bench-write <RBD image name>   * --io-size：单位 byte，默认 4M * --io-threads：线程数，默认 16 * --io-total：总写入字节，默认 1024M * --io-pattern <seq|rand>：写模式，默认为 seq 即顺序写 | * 只能对块设备做写测试 |
| fio + rbd ioengine | fio 结合 rbd IO 引擎的性能测试工具 | 参考 fio --help | * Linux 平台上做 IO 性能测试的瑞士军刀 * 可以对使用内核内 rbd 和用户空间 librados 进行比较 * 标准规则 - 顺序和随机 IO * 块大小 - 4k，16k，64k，256k * 模式 - 读和写 * 支持混合模式 |
| fio + libaio | fio 结合 linux aio 的 rbd 性能测试 |

**3.2 测试结果比较**

* 所使用的命令：
  + rbd bench-write pool.host/image.ph2  --io-total 1719973000 --io-size 4096000 --io-threads 1 --io-pattern rand/seq
  + rados -p pool.host bench 20 write -t 1 --no-cleanup
  + rados -p pool100 load-gen --read-percent 0 --min-object-size 1073741824 --max-object-size 1073741824 --max-ops 1 --read-percent 0/100 --min-op-len 4194304 --max-op-len 4194304 --target-throughput 1073741824 --max\_backlog 1073741824
  + ceph tell osd.0 bench
  + fio/fio -filename=/dev/rbd4 -direct=1 -iodepth 1 -thread -rw=write/read/randwrite/randread -ioengine=libaio -bs=4M -size=1G -numjobs=1 -runtime=120 -group\_reporting -name=read-libaio
* 结果（仅在作者的测试环境的客户端节点上运行以上命令的输出）：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **操作** | **dd 一个 OSD** | **dd 两个 OSD** | **rados load-gen** | **rados bench** | **rbd bench-write** | **ceph tell osd.0 bench** | **fio + rbd** | **fio + libaio** |
| 顺序写 | 165 | 18 | 18 | 18 | 74 MB/s （IOPS 9） | 40 MB/s | 21 （iops 5） | 18（iops 4） |
| 随机写 |  |  |  |  | 67.8 MB/s （IOPS 8） |  | 19 （iops 4） | 16（iops 4） |
| 顺序读 | 460 | 130 | 100 | 109 | N/A |  | 111（iops 27） | 111（iops 27） |
| 随机读 |  |  |  | 112 | N/A |  | 115（iops 28） | 128（iops 31） |

* 简单结论（由于环境、测试方法和数据有限，这些结论不一定正确，有些只是猜测，需要进一步研究，仅供参考）：
  + rados bench 和在两个 OSD 上同时做 dd 的性能差不多。
  + fio + rbd 和 fio + libaio 的结果差不多，相比之下 fio + rbd 还要好一点点。
  + fio 顺序写和读的 BW 和两个 OSD 同时写和读的 BW 差不多。
  + fio 顺序写的 BW 差不多是 单个 OSD 的 bench 的一半 （因为我的 pool 的 size 为 2）。
  + rados load-gen，rodos bench 和 fio rbd/libaio 的结果都差不多，可见都可以信任，只是每一种都有其特长，选择合适你的测试应用场景的某个即可。
  + rdb bench-write 的值明显偏高，原因未知，也许存在 bug，详情可参考 [rbd bench-write vs dd performance confusion](http://www.spinics.net/lists/ceph-users/msg24647.html)，选择时需慎重。

**注意：作者会保持对本文不断更新。**