Hardware configurations

- 4 Dell R620 (the 4 Del R720 had failures controller batteries).
- Single spins working as OSDs with jornals in a separate OSDs partition. From our previous study, journals in SSDs improve writing performance but not so much that justifies buying SSDs for all servers
- Each server has 8 OSDs (we just configured 8 in each storage box leaving 4 disks as spares) giving a total of 32 OSDs

Ceph Object Storage cluster configuration

- Global configuration: Decreased the 'osd map cache size' from the default value of 500 to 100 to limit the memory usage of OSD daemons.

```
auth_service_required = cephx
filestore_xattr_use_omap = true
auth_client_required = cephx
auth_cluster_required = cephx
mon_host = 192.231.127.8,192.231.127.34,192.231.127.26
mon_initial_members = rccephmon1, rccephmon2, rccephmon3
fsid = eea8578f-b3ac-4dfb-a0c5-da40509f5cdc
public network = 192.231.127.0/24
cluster network = 10.100.1.0/24
osd journal size = 10000
osd pool default size = 3
osd pool default min size = 2
osd pool default pg num = 4096
osd pool default pgp num = 4096
osd crush chooseleaf type = 1
osd map cache size = 100
```

- Data / metadata pool configurations:
  - the number of pgs had to be tuned according to the number of OSDs;
  - the pools were configured as normal replication pools

```
### pool: cephfs_dt ###
size: 3
min_size: 2
pg_num: 1024
pgp_num: 1024

### pool: cephfs_mt ###
size: 3
min_size: 2
pg_num: 1024
pgp_num: 1024
```

CephFS configuration and layout

- Using the Ceph kernel module in the client (I had to install kernel-lt-3.10.84-1.el6.elrepo.x86_64.rpm)

- Created several dirs in the /cephfs filesystem, and set up the proper layouts. The different layouts are defined and displayed using extended attributes. All files created under those directories inherit that profile.

  ```
  # getfattr -n ceph.dir.layout objectsize4M_stripeunit64K_stripecount8
  ceph.dir.layout="stripe_unit=65536 stripe_count=8 object_size=4194304 pool=cephfs_dt"
  ```

- The following profiles were defined:

Object Size	**Stripe Unit**	**Stripe Count**
4M	64K	8
	128K	2,4,6,8
	256K	2,4,6,8
	512K	2,4,6,8
512K	64K	2
	128K	2
	256K	2

FIO Benchmark Procedure

- Single thread tests:
  - The fio benchmark was to probe 4 different kinds of operations: sequential write, sequential read, random write and random read.
  - The layout for each fio execution is the following: files of 8 GB; ioengine=libaio; iodepth=64; direct=1.
  - For each type of operation, 6 different values for the fio blocksize were probed: 4k, 64k, 128K, 256k, 512k, 4096k (this represents how the application manages io operations in terms of data chunks)
  - All fio operations were executed for each one of the 16 CephFS layouts depicted above. This results in a grant total of 384 fio tests.

- Multiple threads tests:
  - The fio benchmark was executed with multiple threads (1, 2, 4, 8, 10, 12, 14, 16, 32, 64 and128) probing 4 different kinds of operations: sequential write, sequential read, random write and random read.
  - The layout for each fio execution is the following: files of 8 GB; ioengine=libaio; iodepth=64; direct=1
  - The CephFS layout was kept constant during all executions (object size (4M), stripe unit (512K) and stripe count (8)).
  - The block size of the fio test was set to 512K (to match the stripe unit).

Analysis

1. Stripe unit effect

Figure 1 shows the results for the fio bandwith measurements as a function of the stripe unit for the CephFS layout. The other parameters of the CephFS layout were kept fixed during the test, namely the object size (4M) and the object count (8). The fio tests were executed in sequential write, random write, sequential read and random read mode, using different block sizes (4K, 64K, 128K, 256K and 512K).

The results show that there is a clear correlation between the stripe unit and the block size used in the fio test. The observed behaviour, for all kind of operation modes, is the following:

1. The bandwith growths for (stripe unit < fio block size)
2. The bandwidth reaches its max for the match (fio blocksize = stripe unit).
3. The bandwidth saturates at (approximately) its maximum value for (stripe unit > fio block size)

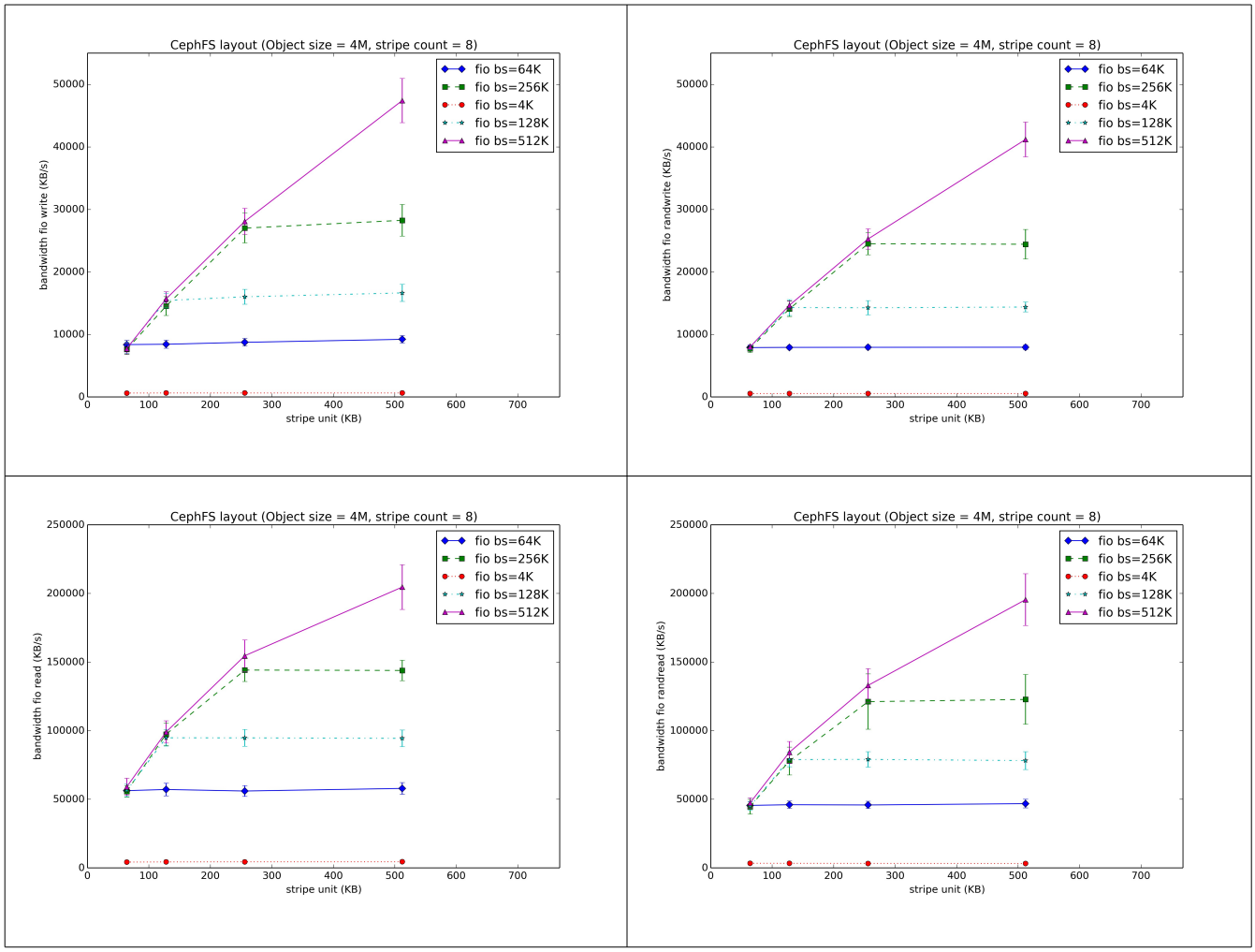

Figure 1: Fio bandwith results as a function of the stripe unit of the CephFS layout. Other CephFS parameters (object_size=4M and stripe_count=8) were kept fixed during the tests. Fio was run with different block size values (4K, 64K, 128K, 256K and 512K) and in 4 different modes

(sequential write (top left), random write (top right), sequential read (bottom left) and random read (bottom right))

The fio block size plays an important role in these tests since it represents how the data is chucked in I/O operations:

- If we deploy CephFS to serve user homes, we know that (in the majority of the cases), users will be dealing with small (a couple of KB) files (txt, code, etc...). To conclude how the system would behave in that case, we should look to the fio results obtained with the smaller blocksizes (4K to 128K). The approximately maximum bandwidths in those cases, obtained for the match (fio blocksize = stripe unit), are:

BW (KB/s)				
Fio BS = Stripe Unit	Seq Write	Rand Write	Seq Read	Rand Read
4 KB (*)	632 ± 46	554 ± 17	4258 ± 407	3376 ± 173
64 KB	8385 ± 663	7905 ± 329	56314 ± 4691	45564 ± 2683
128 KB	15434 ± 1174	14312 ± 1263	94834 ± 6114	78976 ± 5236

- If we deploy CephFS to serve data files, we know that it will be used to store big files (several MB / GB). To conclude how the system would behave in that case, we should look to the fio results obtained with the higher blocksizes (512K). If we actually knew the exact use cases, and were able to know the file sizes, we could even increase the stripe_unit. The approximately maximum bandwidths obtained for the match (fio blocksize = stripe unit = 512K) are:

BW (KB/s)				
Fio BS = Stripe Unit	Seq Write	Rand Write	Seq Read	Rand Read
512 KB	47420 ± 3528	41197 ± 2773	204667 ± 16257	195492 ± 18859

2. Stripe count effect

Figure 2 shows the results for the fio bandwith measurements as a function of the stripe count for the CephFS layout. The other parameters of the CephFS layout were kept fixed during the test: the object size (4M) and the stripe unit (512K). The fio tests were executed in sequential write, random write, sequential read and random read mode using different block sizes (4K, 64K, 128K, 256K and 512K).

According to the documentation, if you anticipate large images sizes, you may see considerable read/write performance improvements by striping client data over multiple objects within an object set. Significant write performance occurs when the client writes the stripe units to their corresponding objects in parallel. Since objects get mapped to different placement groups and further mapped to different OSDs, each write occurs in parallel at the maximum write speed. A write to a

single disk would be limited by the head movement (e.g. 6ms per seek) and bandwidth of that one device (e.g. 100MB/s). By spreading that write over multiple objects (which map to different placement groups and OSDs) Ceph can reduce the number of seeks per drive and combine the throughput of multiple drives to achieve much faster write (or read) speeds.

The results we obtained do not show a dependency of the bandwidth values as a function of the stripe count. The conclusions are similar if we perform the same analysis for a different CephFS layout in terms of stripe unit (128K or 256K). The difference may only be visible for the manipulation of bigger files (we were using 8GB files) or in the increase of concurrent accesses, running the test with multiple threads.

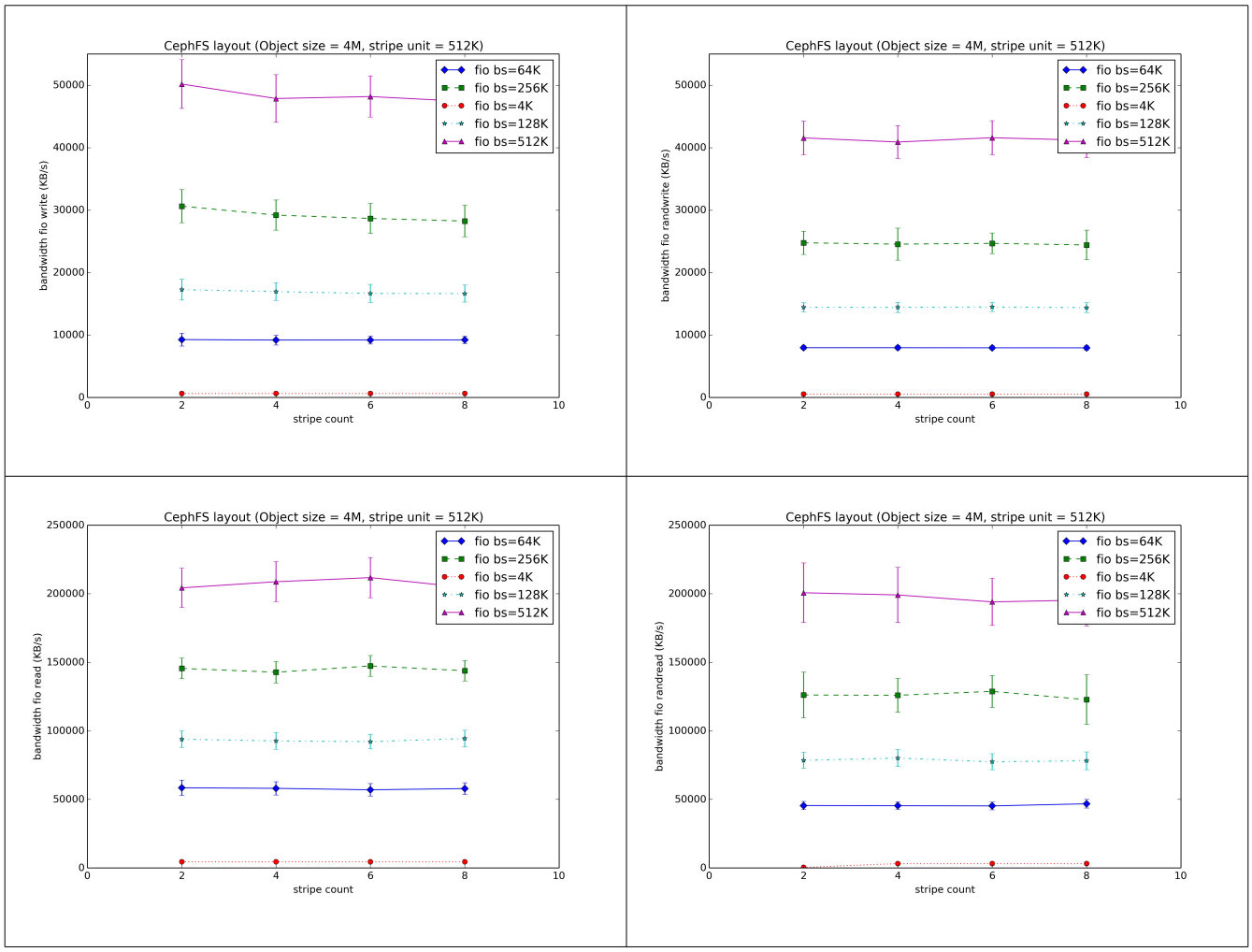

Figure 2: Fio bandwith results as a function of the stripe count of the CephFS layout. Other CephFS parameters (object_size=4M and stripe_unit=512K) were kept fixed during the tests. Fio was run with different block size values (4K, 64K, 128K, 256K and 512K) and in 4 different modes (sequential write (top left), random write (top right), sequential read (bottom left) and random read (bottom right))

Object Size effect

The previous set of benchmarks to the Ceph Storage cluster using rados client showed that the best performance was achieved for big object sizes. That was why the majority of the tests was executed with an object size of 4M.

Nevertheless, we have tried to reassess that conclusion by execution different fio operation in a ceph directory with a layout with a different object size. The following table provides quantitative measurements of this analysis. The results confirm that a bigger object size improve bandwidth performance, specially for read operations. However, the price to pay may be a higher occupation of the file system since, in the case of small files, objects may not be totally filled up.

BW (KB/s)					
FioBS=StripeUnit	**Obj Size**	**Seq Write**	**Rand Write**	**Seq Read**	**Rand Read**
128K	4M	15434 ± 1174	14327 ± 1263	101018 ± 737	77583 ± 5814
	512K	15047 ± 802	13535 ± 1092	68586 ± 2617	47397 ± 3974
256K	4M	27016 ± 2385	24828 ± 1745	148953 ± 9306	129020 ± 11440
	512K	24021 ± 1946	22760 ± 1652	91428 ± 4150	78092 ± 5052

Number of Threads effect

Figure 3 shows the results for the aggregated fio bandwith measurements as a function of the number of threads. Figure 4 shows the bandwidth experienced by each thread operation, also as a function of the number of threads. The CephFS layout was kept fixed during fio tests execution: object size (4M), stripe unit (512K) and stripe count (8). The block size of the fio tests was set equal to the stripe unit (512K) since previous tests have shows that this is the setting that maximizes the bandwidth.

The results shows that the aggregated bandwidth scales as a function of the number of threads. Up to 128 threads, the maximum value tested, the results for the aggregated bandwidth do not show signs to saturate. In the read case, it seems to be almost hitting the theoretical value of 1.25 GB/s. Curiously, when the number of threads increases, the random operations perform better than the sequential ones.

Agreggated BW (KB/s)				
Nthreads	Seq Write	Rand Write	Seq Read	Rand Read
128	232889	270319	929333	956799

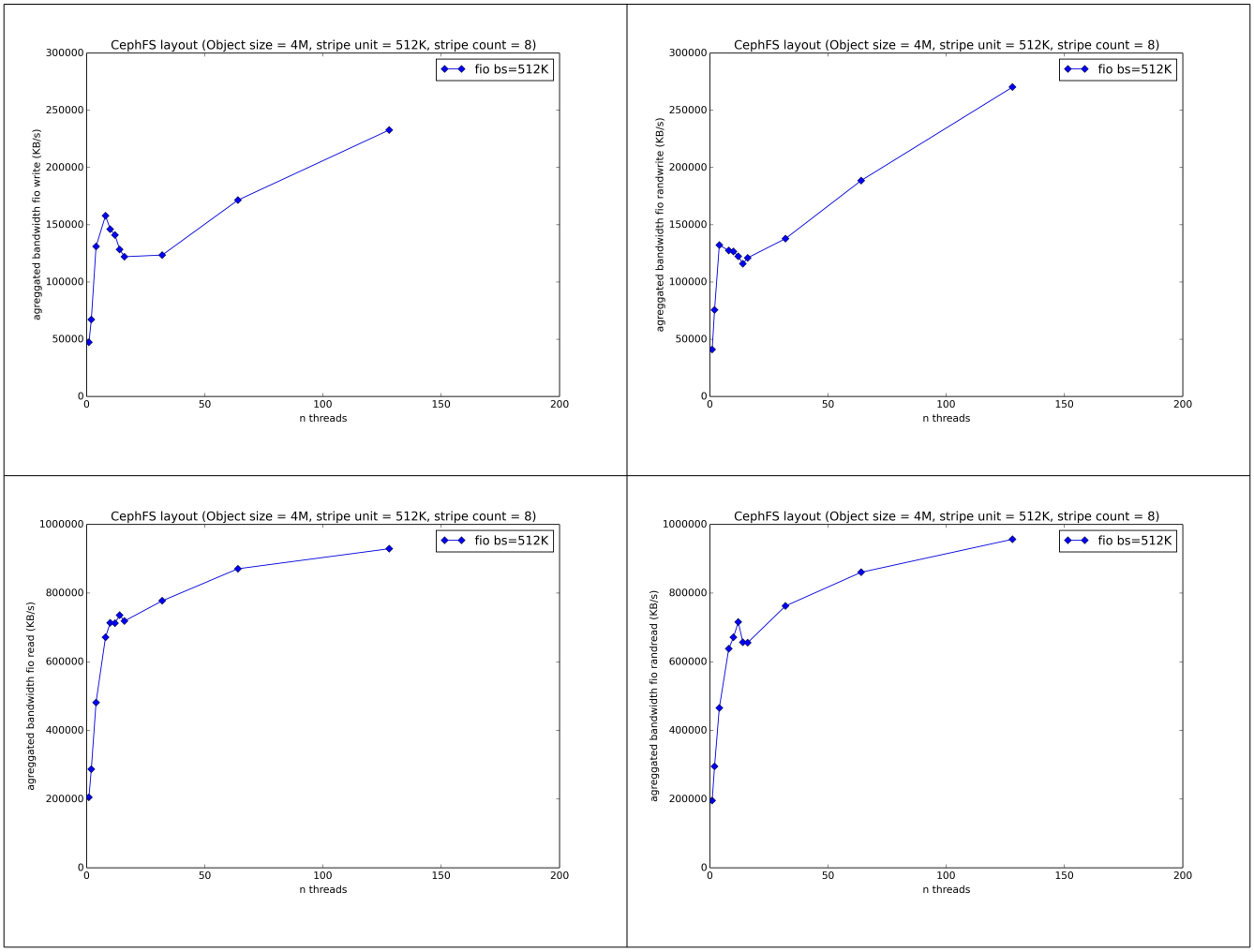

Figure 3: Aggregated bandwidth as a function of the number of threads. The CephFS layout was kept fixed namely object size (4M), stripe unit (512K) and stripe count (8). The block size of the fio tests was set to match the stripe unit (512K), and the fio tests were executed for sequential write (top left), random write (top right), sequential read (bottom left) and random write (bottom right) operations.

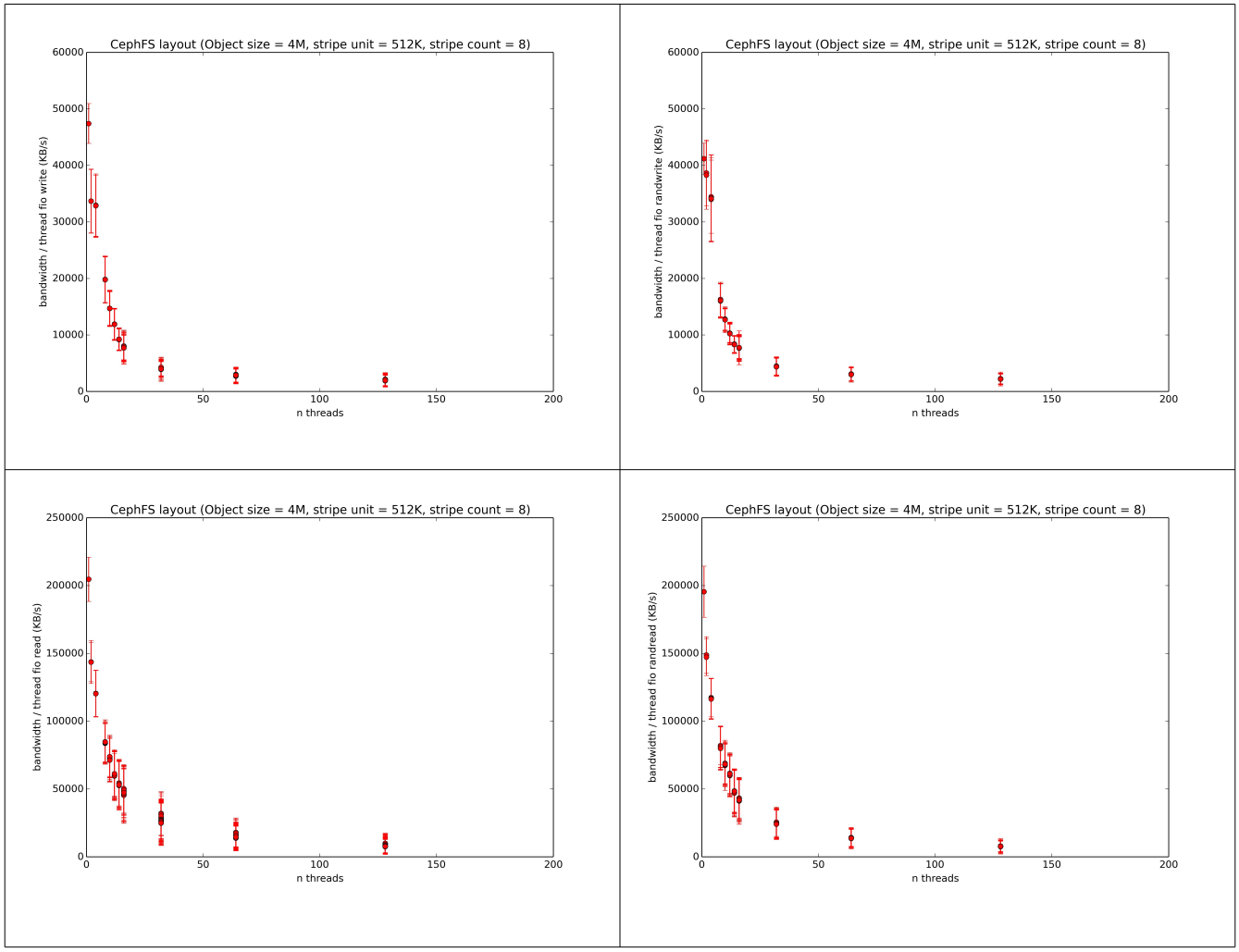

Figure 4: Bandwidth / thread as a function of the number of threads. The CephFS layout was kept fixed namely object size (4M), stripe unit (512K) and stripe count (8). The block size of the fio tests was set to match the stripe unit (512K), and the fio tests were executed for sequential write (top left), random write (top right), sequential read (bottom left) and random write (bottom right) operations.

Conclusions

1. The stripe unit is the most important parameter for performance optimization. The stripe unit should be tuned for the specific usage use case. The results point to the fact that best performance is achieve for bigger stripe units. However, there is the constrain that the stripe unit should be lower than the object size, and if we set the stripe unit too big, the object size also has to be bigger. As a consequence,vthe minimum size occupied by any files in the filesystem automatically grows.
2. Our tests did not showed a significant dependence of the bandwidth values as a function of the stripe count. The effect may only be visible for the manipulation of bigger files (we were using 8GB files) or in the increase of concurrent accesses, running the test with multiple threads. Nevertheless, the documentation strongly suggests to stripe data across multiple objects, and we should follow the remark.

3. Bigger object size are more beneficial for performance, specially for read operations. However, the price to pay may be a higher occupation of the file system since, in the case of small files, objects may not be totally filled up.
4. Ceph / CephFS performance scales as a function of the number of threads. Up to 128 threads, the maximum value tested, the results for the aggregated bandwidth do not show signs to saturate. In the read case, it seems to be almost hitting the theoretical value of 1.25 GB/s.