



New Read Balancer in Ceph

Ceph Virtual 2022

Josh Salomon and Laura Flores



- In a distributed storage system like Ceph, it is important to balance write and read requests across OSDs for optimal performance.
- The existing capacity balancer works well to balance write requests, but there is still a need to balance read requests, especially in small clusters and pools with less PGs.
- In order to improve the Ceph balancing process, we established three goals:
 1. Collect osdmaps from the Ceph community, particularly from small clusters, for better testing
 2. Refactor the existing balancer code (`calc_pg_upmaps`) to make it easier for Ceph developers to understand the code and contribute
 3. Implement a workload balancer to balance read requests on a pool-by-pool basis



- **A Functional requirement**
 - A cluster is as full as the fullest device
 - Therefore it is a strict requirement
- **Balancing is expensive**
 - It takes time, and during this time the performance of the system is reduced.
 - It requires data movement (by definition)
- **It balances the write performance ...**
 - ... if all devices are homogenous (same size and performance)

Read Balancer (a.k.a Primary Balancer)



- A Performance requirement
 - Unbalanced reads reduce the cluster overall bandwidth (due to weakest link in the chain effect)
- Balancing is cheap
 - It is just a metadata operation, fast and involve no data movement
 - No impact on the cluster performance (except improved performance when the operation completes (almost immediately))
- It balances the read performance ...
 - ... if all devices are homogenous (same size and performance)
- BUT - in future versions
 - The same mechanism can be used to improve overall cluster performance in heterogeneous systems
 - The same mechanism can be used to compensate on node performance fluctuations



- **Crush primary balancing (on homogeneous systems):**
 - No active balancing code
 - Crush random distribution improves with larger PGs number
 - In general larger systems are quite balanced, smaller system may not be balanced
 - Most ODF instances are small, homogenous clusters.
- **On heterogeneous systems:**
 - Smaller devices get smaller loads (may fit some EBS pricing schemes, but not HDDs or SSDs)
 - Larger systems tend to become heterogenous over time
- **Conclusion:**
 - Read balancer (the Reef feature) is useful mostly for ODF clusters
 - Workload Balancer (planned for future versions, will be based on the read balancer infrastructure) will be useful for large, heterogeneous clusters

Stepping Stones 1/2



Read balancer score:

Added additional item to the output of the command: `ceph osd pool ls detail`

(applicable only for replicated pool)

```
[jsalomon@Josh-laptop build$] ./bin/ceph osd pool ls detail 2>/dev/null
pool 1 '.rgw.root' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 16 autoscale_mode on last_change 46 lfor 0/0/34 flags hashspool stripe_width 0 application rgw read_balance_score 2.00
pool 2 'default.rgw.log' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 4 pgp_num_target 16 autoscale_mode on last_change 59 lfor 0/0/36 flags hashspool stripe_width 0 application rgw read_balance_score 3.00
pool 3 'default.rgw.control' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 1 pgp_num 1 autoscale_mode on last_change 25 flags hashspool stripe_width 0 application rgw read_balance_score 3.00
pool 4 'default.rgw.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 4 pgp_num_target 16 autoscale_mode on last_change 59 lfor 0/0/38 flags hashspool stripe_width 0 pg_autoscale_bias 4 application rgw read_balance_score 1.00
pool 5 'default.rgw.buckets.index' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 16 autoscale_mode on last_change 46 lfor 0/0/40 flags hashspool stripe_width 0 pg_autoscale_bias 4 application rgw read_balance_score 1.25
pool 6 'default.rgw.buckets.data' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 16 autoscale_mode on last_change 46 lfor 0/0/42 flags hashspool stripe_width 0 application rgw read_balance_score 1.50
pool 7 'ecp' erasure profile default size 3 min_size 2 crush_rule 1 object_hash rjenkins pg_num 16 pgp_num 16 autoscale_mode on last_change 49 flags hashspool stripe_width 8192
```

Read Balance Score



The meaning of the read balance score:

- Score of 1.0 is optimal
 - In most cases we can't achieve this score
 - When all primaries are in the same OSD the score is the replica count
- All scores in the range [1.0, replica_count]
- Score of $1 + x$ shows approximately $(x*100)\%$ degradation in pool read performance under load
 - Score of 1.2 is roughly 20% degradation.

Read Balance Score - more info I



```
[jsalomon@Josh-laptop build$] ./bin/ceph osd pool ls detail -f json-pretty | jq '.[0] | select(.pool == 6)'
```

... skipped most of the json output here ...

```
"application_metadata": {  
  "rgw": {}  
},  
"read_balance": {  
  "score_acting": 1.5,  
  "score_stable": 1.5,  
  "optimal_score": 1,  
  "raw_score_acting": 1.5,  
  "raw_score_stable": 1.5,  
  "primary_affinity_weighted": 1,  
  "average_primary_affinity": 1,  
  "average_primary_affinity_weighted": 1  
}
```

⇐ used to be the last object in the json/xml file

⇐ A new object with more information on read balance score

⇐ scores calculated on primaries as well as acting primaries

⇐ optimal score may change when we have devices with low primary affinity

⇐ We have information about the score calculation before and after adjustments to the primary affinity and its weights

Read Balance Score - more info II



Reducing primary affinity of some OSDs

```
[jsalomon@Josh-laptop build$]./bin/ceph osd pool ls detail -f json-pretty | jq '.[0] | select(.pool == 6)'
```

... skipped most of the json output here ...

```
"application_metadata": {  
  "rgw": {}  
},  
"read_balance": {  
  "score_acting": 1.2687473297119141,  
  "score_stable": 1.2687473297119141,  
  "optimal_score": 1.3793132305145264,  
  "raw_score_acting": 1.75,  
  "raw_score_stable": 1.75,  
  "primary_affinity_weighted": 0.72499847412109375,  
  "average_primary_affinity": 0.72499847412109375,  
  "average_primary_affinity_weighted": 0.72499847412109375  
}
```

Score = raw_score / optimal_score

Weighted info takes into account the OSD sizes, in this case all OSDs are in the same size hence average_primary_affinity and average_primary_affinity_weighted are identical.

Stepping Stones 2/2



Added two commands to change the primary OSD for a PG in a replicated pool. The new primary must be one of the OSDs which participate in this PG.

- `ceph osd pg-upmap-primary <pgid> <osdid>`
- `ceph osd rm-pg-upmap-primary <pgid>`

(These commands are applicable only for replicated pools.)

```
[jsalomon@Josh-laptop build$] ./bin/ceph pg dump pgs_brief | grep "6\.1"
```

```
dumped pgs_brief
```

```
6.1 active+clean [1,3,2] 1 [1,3,2] 1
```

```
[jsalomon@Josh-laptop build$] ./bin/ceph osd pg-upmap-primary 6.1 3
```

```
change primary for pg 6.1 to osd.3
```

```
[jsalomon@Josh-laptop build$] ./bin/ceph pg dump pgs_brief | grep "6\.1"
```

```
dumped pgs_brief
```

```
6.1 active+clean [3,1,2] 3 [3,1,2] 3
```



Two important functions make up the read balancer's framework:

1. **OSDMap::calc_desired_primary_distribution**
 - a. A policy function that can be changed
2. **OSDMap::balance primaries**
 - a. The overall balancing algorithm

A deeper dive into OSDMap::calc_desired_primary_distribution



What does this function do?

Overall, it **calculates the optimal amount of primary pgs** that should be on each OSD per pool. It does so by:

1. Checking that the pool we're balancing is **replicated**
2. Finding the **replica count** of the pool
3. Finding the **num_pgs** on each OSD
4. Calculating the **desired_primary_distribution** for each OSD, which is:
 - a. **$\text{desired_primary_distribution} = (\text{num_pgs} / \text{replica_count}) * \text{primary_affinity}(\text{OSD})$**
5. Stretching the **desired_primary_distribution** for each OSD, so total distribution = the total number of pgs in the cluster. This applies mainly when primary affinity is smaller than 1.

A deeper dive into OSDMap::balance_primaries



What does this function do?

This is the function that actually **does the read balancing**. It does so by:

1. Using OSDMap::calc_desired_primary_distribution to find the **optimal primary distribution** for each OSD on the pool we want to balance
2. Calculating **how much each OSD deviates** from its optimal primary distribution
3. Based on the above deviations, **swapping PGs** so each OSD gets as close to its optimal primary distribution as possible
 - i. This step ensures that primaries are only swapped between OSDs on the same PG so there is no data movement
4. Returning the **new pg mappings** and **total number of changes**

The future of these implementations



How might we enhance these functions in the future?

`OSDMap::calc_desired_primary_distribution`

1. The current implementation can be thought of as “phase 1”, since it assumes all devices are identical.
2. Future improvements may involve implementing a policy that puts more read load on smaller disks, which are known to have less write load.

`OSDMap::balance primaries`

1. Aside from any optimizations, the basic structure of this function will not change.
2. It can be thought of as a base tree trunk that may grow more branches, or optimizations, in the future.



- **Demo 1:** Balance a cluster with devices of equal primary affinity
- **Demo 2:** Balance a cluster where one device has 0 primary affinity

*** Notes ***

- Since the new “pg-upmap-primary” commands are not merged yet, we use “primary-temp” command for this demo.
- Screenshots of this demo are available at the end of the presentation.

Future Plans and Improvements



- Conduct performance tests
- Add read balancer integration tests to the teuthology rados suite
- Turn the read balancer on by default as part of the balancer manager module
- Account for devices of different sizes



Questions



Demo 1:

Balance a cluster with devices of equal primary affinity

Demo 1



Check ceph status

```
$ ./bin/ceph -s

cluster:
  id:      72320b2f-510f-49ee-91d3-2ba52bbe4912
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum a,b,c (age 4m)
  mgr: x(active, since 4m)
  mds: 1/1 daemons up, 2 standby
  osd: 4 osds: 4 up (since 3m), 4 in (since 4m)
  rgw: 2 daemons active (1 hosts, 1 zones)

data:
  volumes: 1/1 healthy
  pools:   7 pools, 184 pgs
  objects: 235 objects, 459 KiB
  usage:   4.0 GiB used, 400 GiB / 404 GiB avail
  pgs:    184 active+clean
```

Demo 1



Check pool details for current read balancer scores. We'll try to improve the score of pool 6, or “default.rgw.control”:

```
$ ./bin/ceph osd pool ls detail
```

```
pool 1 '.mgr' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 8 pgp_num 8 autoscale_mode on last_change 35 lfor 0/0/33 flags hashspool stripe_width 0 pg_num_max 32 pg_num_min 8 application mgr read_balance_score 1.50
pool 2 'cephfs.a.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 16 autoscale_mode on last_change 114 lfor 0/0/39 flags hashspool stripe_width 0 pg_autoscale_bias 4 pg_num_min 16 recovery_priority 5 application cephfs read_balance_score 1.50
pool 3 'cephfs.a.data' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 45 lfor 0/0/39 flags hashspool stripe_width 0 application cephfs read_balance_score 1.62
pool 4 '.rgw.root' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 45 lfor 0/0/41 flags hashspool stripe_width 0 application rgw read_balance_score 1.38
pool 5 'default.rgw.log' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 165 lfor 0/0/41 flags hashspool stripe_width 0 application rgw read_balance_score 1.25
pool 6 'default.rgw.control' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 45 lfor 0/0/43 flags hashspool stripe_width 0 application rgw read_balance_score 1.62
pool 7 'default.rgw.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 45 lfor 0/0/43 flags hashspool stripe_width 0 pg_autoscale_bias 4 application rgw read_balance_score 1.25
```

Demo 1



Get latest copy of your OSD map

```
$ ./bin/ceph osd getmap -o om  
got osdmap epoch 169
```

Demo 1



Run the upmap balancer first to make sure writes are balanced. In this case, the upmap balancer was unable to optimize further.

```
$ ./bin/osdmactool om --upmap out.txt

./bin/osdmactool: osdmap file 'om'
writing upmap command output to: out.txt
checking for upmap cleanups
upmap, max-count 10, max deviation 5
pools cephfs.a.meta .rgw.root cephfs.a.data default.rgw.log .mgr default.rgw.control default.rgw.meta
prepared 0/10 changes
Unable to find further optimization, or distribution is already perfect
```

Demo 1



Run the read balancer,
focusing on
“default.rgw.control”.

```
$ ./bin/osdmaptool om --vstart --read out.txt --read-pool default.rgw.control
```

```
./bin/osdmaptool: osdmap file 'om'  
writing upmap command output to: out.txt
```

```
----- BEFORE -----
```

osd.0		primary affinity: 1		number of prims: 10
osd.1		primary affinity: 1		number of prims: 6
osd.2		primary affinity: 1		number of prims: 3
osd.3		primary affinity: 1		number of prims: 13

```
read_balance_score of 'default.rgw.control': 1.625
```

```
----- AFTER -----
```

osd.0		primary affinity: 1		number of prims: 8
osd.1		primary affinity: 1		number of prims: 8
osd.2		primary affinity: 1		number of prims: 8
osd.3		primary affinity: 1		number of prims: 8

```
read_balance_score of 'default.rgw.control': 1
```

```
num changes: 9
```

Demo 1



We can check to see what the balancer suggests in the “out.txt” file:

```
$ cat out.txt

./bin/ceph osd primary-temp 6.0 2
./bin/ceph osd primary-temp 6.2 2
./bin/ceph osd primary-temp 6.3 2
./bin/ceph osd primary-temp 6.5 1
./bin/ceph osd primary-temp 6.6 2
./bin/ceph osd primary-temp 6.7 1
./bin/ceph osd primary-temp 6.8 0
./bin/ceph osd primary-temp 6.9 2
./bin/ceph osd primary-temp 6.b 0
```


Demo 1



We can apply the file to a live system if we choose:

```
$ source out.txt
```

```
set 6.0 primary_temp mapping to 2
set 6.2 primary_temp mapping to 2
set 6.3 primary_temp mapping to 2
set 6.5 primary_temp mapping to 1
set 6.6 primary_temp mapping to 2
set 6.7 primary_temp mapping to 1
set 6.8 primary_temp mapping to 0
set 6.9 primary_temp mapping to 2
set 6.b primary_temp mapping to 0
```

Demo 1



Notice how the score has improved to 1.00 for “default.rgw.control”, which previously had a score of 1.62:

```
$ ./bin/ceph osd pool ls detail
```

```
pool 1 'mgr' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 8 pgp_num 8 autoscale_mode on last_change 35 lfor 0/0/33 flags hashspool stripe_width 0 pg_num_max 32 pg_num_min 8 application mgr read_balance_score 1.50
pool 2 'cephfs.a.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 16 autoscale_mode on last_change 114 lfor 0/0/39 flags hashspool stripe_width 0 pg_autoscale_bias 4 pg_num_min 16 recovery_priority 5 application cephfs read_balance_score 1.50
pool 3 'cephfs.a.data' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 45 lfor 0/0/39 flags hashspool stripe_width 0 application cephfs read_balance_score 1.62
pool 4 '.rgw.root' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 45 lfor 0/0/41 flags hashspool stripe_width 0 application rgw read_balance_score 1.38
pool 5 'default.rgw.log' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 165 lfor 0/0/41 flags hashspool stripe_width 0 application rgw read_balance_score 1.25
pool 6 'default.rgw.control' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 45 lfor 0/0/43 flags hashspool stripe_width 0 application rgw read_balance_score 1.00
pool 7 'default.rgw.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 45 lfor 0/0/43 flags hashspool stripe_width 0 pg_autoscale_bias 4 application rgw read_balance_score 1.25
```



Demo 2:

Balance a cluster where one device has 0 primary affinity

Demo 2



Check ceph status

```
$ ./bin/ceph -s

cluster:
  id:      5b0445f5-b8af-4c3a-b3b8-bf2a443657f6
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum a,b,c (age 5m)
  mgr: x(active, since 5m)
  mds: 1/1 daemons up, 2 standby
  osd: 4 osds: 4 up (since 5m), 4 in (since 5m)
  rgw: 2 daemons active (1 hosts, 1 zones)

data:
  volumes: 1/1 healthy
  pools:   7 pools, 184 pgs
  objects: 232 objects, 458 KiB
  usage:   4.0 GiB used, 400 GiB / 404 GiB avail
  pgs:    184 active+clean
```

Demo 2



Change primary affinity on one of the OSDs.

```
$ ./bin/ceph osd primary-affinity 2 0  
set osd.2 primary-affinity to 0 (802)
```

Demo 2



Check that no primary PGs are on osd.2.

```
$ ./bin/ceph pg dump pgs_brief | grep "6\."
```

6.1a	active+clean	[3,0,1]	3	[3,0,1]	3
6.1b	active+clean	[3,2,1]	3	[3,2,1]	3
6.18	active+clean	[0,1,3]	0	[0,1,3]	0
6.19	active+clean	[0,1,3]	0	[0,1,3]	0
6.1e	active+clean	[3,2,0]	3	[3,2,0]	3
6.1f	active+clean	[3,0,2]	3	[3,0,2]	3
6.1c	active+clean	[3,2,1]	3	[3,2,1]	3
6.1d	active+clean	[1,0,2]	1	[1,0,2]	1
6.12	active+clean	[0,2,1]	0	[0,2,1]	0
6.13	active+clean	[3,0,2]	3	[3,0,2]	3
6.10	active+clean	[0,2,1]	0	[0,2,1]	0
6.11	active+clean	[3,0,1]	3	[3,0,1]	3
6.16	active+clean	[0,1,3]	0	[0,1,3]	0
6.17	active+clean	[1,2,3]	1	[1,2,3]	1
6.14	active+clean	[3,2,1]	3	[3,2,1]	3
6.15	active+clean	[0,2,1]	0	[0,2,1]	0
6.6	active+clean	[3,0,2]	3	[3,0,2]	3
6.1	active+clean	[1,3,2]	1	[1,3,2]	1
6.4	active+clean	[1,3,2]	1	[1,3,2]	1
6.7	active+clean	[3,2,1]	3	[3,2,1]	3
6.5	active+clean	[0,1,3]	0	[0,1,3]	0
6.2	active+clean	[3,2,0]	3	[3,2,0]	3
6.3	active+clean	[0,2,3]	0	[0,2,3]	0
6.0	active+clean	[0,3,2]	0	[0,3,2]	0
6.c	active+clean	[3,1,2]	3	[3,1,2]	3
6.d	active+clean	[1,0,2]	1	[1,0,2]	1
6.e	active+clean	[1,2,3]	1	[1,2,3]	1
6.f	active+clean	[3,2,1]	3	[3,2,1]	3
6.8	active+clean	[3,2,0]	3	[3,2,0]	3
6.9	active+clean	[0,1,2]	0	[0,1,2]	0
6.a	active+clean	[0,1,2]	0	[0,1,2]	0
6.b	active+clean	[3,1,0]	3	[3,1,0]	3

Demo 2



Check pool details for current read balancer scores. We'll try to improve the score of pool 6, or “default.rgw.control”:

```
$ ./bin/ceph osd pool ls detail
```

```
pool 1 'mgr' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 8 pgp_num 8 autoscale_mode on last_change 37 lfor 0/0/35 flags hashpspool stripe_width 0 pg_num_max 32 pg_num_min 8 application mgr read_balance_score 1.50
pool 2 'cephfs.a.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 16 autoscale_mode on last_change 115 lfor 0/0/41 flags hashpspool stripe_width 0 pg_autoscale_bias 4 pg_num_min 16 recovery_priority 5 application cephfs read_balance_score 1.50
pool 3 'cephfs.a.data' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 47 lfor 0/0/41 flags hashpspool stripe_width 0 application cephfs read_balance_score 1.41
pool 4 'rgw.root' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 47 lfor 0/0/43 flags hashpspool stripe_width 0 application rgw read_balance_score 1.41
pool 5 'default.rgw.log' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 155 lfor 0/0/43 flags hashpspool stripe_width 0 application rgw read_balance_score 1.03
pool 6 'default.rgw.control' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 47 lfor 0/0/45 flags hashpspool stripe_width 0 application rgw read_balance_score 1.41
pool 7 'default.rgw.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 47 lfor 0/0/45 flags hashpspool stripe_width 0 pg_autoscale_bias 4 application rgw read_balance_score 1.12
```

Demo 2



Get latest copy of your OSD map.

```
$ ./bin/ceph osd getmap -o om  
got osdmap epoch 161
```


Demo 2



Run the upmap balancer first to make sure writes are balanced. In this case, the upmap balancer was unable to optimize further.

```
$ ./bin/osdmapprool om --upmap out.txt  
./bin/osdmapprool: osdmap file 'om'  
writing upmap command output to: out.txt  
checking for upmap cleanups  
upmap, max-count 10, max deviation 5  
pools default.rgw.log cephfs.a.data default.rgw.control default.rgw.meta .mgr cephfs.a.meta .rgw.root  
prepared 0/10 changes  
Unable to find further optimization, or distribution is already perfect
```

Demo 2



Run the read balancer,
focusing on
“default.rgw.control”:

```
$ ./bin/osdmactool om --vstart --read out.txt --read-pool default.rgw.control
./bin/osdmactool: osdmap file 'om'
writing upmap command output to: out.txt

----- BEFORE -----
osd.0 | primary affinity: 1 | number of prims: 11
osd.1 | primary affinity: 1 | number of prims: 6
osd.3 | primary affinity: 1 | number of prims: 15
read_balance_score of 'default.rgw.control': 1.40625

----- AFTER -----
osd.0 | primary affinity: 1 | number of prims: 10
osd.1 | primary affinity: 1 | number of prims: 11
osd.3 | primary affinity: 1 | number of prims: 11
read_balance_score of 'default.rgw.control': 1.03125

num changes: 6
```

Demo 2



We can check to see what the balancer suggests in the “out.txt” file:

```
$ cat out.txt

./bin/ceph osd primary-temp 6.5 1
./bin/ceph osd primary-temp 6.7 1
./bin/ceph osd primary-temp 6.9 1
./bin/ceph osd primary-temp 6.b 0
./bin/ceph osd primary-temp 6.c 1
./bin/ceph osd primary-temp 6.f 1
```

Demo 2



We can apply the file to a live system if we choose:

```
$ source out.txt
```

```
set 6.5 primary_temp mapping to 1  
set 6.7 primary_temp mapping to 1  
set 6.9 primary_temp mapping to 1  
set 6.b primary_temp mapping to 0  
set 6.c primary_temp mapping to 1  
set 6.f primary_temp mapping to 1
```

Demo 2



Notice how the score has improved to 1.03 for “default.rgw.control”, which previously had a score of 1.41:

```
$ ./bin/ceph osd pool ls detail
```

```
pool 1 'mgr' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 8 pgp_num 8 autoscale_mode on last_change 37 lfor 0/0/35 flags hashspool stripe_width 0 pg_num_max 32 pg_num_min 8 application mgr read_balance_score 1.50
pool 2 'cephfs.a.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 16 pgp_num 16 autoscale_mode on last_change 115 lfor 0/0/41 flags hashspool stripe_width 0 pg_autoscale_bias 4 pg_num_min 16 recovery_priority 5 application cephfs read_balance_score 1.50
pool 3 'cephfs.a.data' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 47 lfor 0/0/41 flags hashspool stripe_width 0 application cephfs read_balance_score 1.41
pool 4 'rgw.root' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 47 lfor 0/0/43 flags hashspool stripe_width 0 application rgw read_balance_score 1.41
pool 5 'default.rgw.log' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 155 lfor 0/0/43 flags hashspool stripe_width 0 application rgw read_balance_score 1.03
pool 6 'default.rgw.control' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 47 lfor 0/0/45 flags hashspool stripe_width 0 application rgw read_balance_score 1.03
pool 7 'default.rgw.meta' replicated size 3 min_size 1 crush_rule 0 object_hash rjenkins pg_num 32 pgp_num 32 autoscale_mode on last_change 47 lfor 0/0/45 flags hashspool stripe_width 0 pg_autoscale_bias 4 application rgw read_balance_score 1.12
```



The scripts used for both demos are available at this link:

<https://gist.github.com/ljflores/171919e940cbf89b283688634317b334>