Ceph Improved Read Balancer

OSD Size aware read balancer

Josh Salomon, Red Hat

Laura Flores, IBM

Red Hat

Motivation

- Capacity balancing is a mandatory functional requirement for software defined storage systems
- This implies that larger devices handle more capacity
- Which means that larger devices are more loaded
- Which implies that under load they become the weakest link in the performance chain
- So larger devices make the system slower
- This is just "the physics laws of distributed loaded systems"

Motivation

- Capacity balancing is a mandatory functional requirement for software defined storage systems
- This implies that larger devices handle more capacity
- Which means that larger devices are more loaded
- Which implies that under load they become the weakest link in the performance chain
- So larger devices make the system slower
- This is just "the physics laws of distributed loaded systems"
- Or possibly not always...

Assumptions

- Most devices bandwidth depends just on the technology and not on the capacity
  - Note: This is incorrect for some AWS EBS types
- Write load is decided based on the PG distribution (which depends on the device capacity)
- Read load depends on the primary distribution (which can be changed as of Reef)
- Smaller devices have less load than larger devices, hence we can give them more read tasks (more primaries)
- This breaks "the physics laws of distributed loaded systems", and gives better cluster performance.

Assumptions

- Most devices bandwidth depends just on the technology and not on the capacity
  - Note: This is incorrect for some AWS EBS types
- Write load is decided based on the PG distribution (which depends on the device capacity)
- Read load depends on the primary distribution (which can be changed as of Reef)
- Smaller devices have less load than larger devices, hence we can give them more read tasks (more primaries)
- This breaks "the physics laws of distributed loaded systems", and gives better cluster performance.
- Up to some level – this is not a magic remedy for every heterogenous OSD configuration

Assumptions

- So we need to assume one thing - what is the read ratio out of all the IOs per pool
- Step 1: a new pool parameter was added `read_ratio`
- Usage:
  - `ceph osd pool set rbd read_ratio 70`
    - Valid values - [0..100].
    - 0 unsets this parameter, 1-100 are the predicted read IOs percentage out of all IOs to this pool
  - `ceph osd pool get rbd read_ratio`
- Applicable only to replicated pool.
- Should not be fully accurate, but should the closer it is to the real value the better the cluster performance will be
- **Note:** Currently the PR does not handle PGs of different weights

Red Hat

System Load Calculation

- First step: PG load
  - The load of a single PG = *100 + (100 - read_ratio)*(pool_size - 1)*
  - For every 100 IOs on the system the primary performs 100 IOs and the secondaries perform only the write IOs
- Pool load = *PG load * PG num*
- Desired load per OSD = *pool load / osd num*

System Load Calculation

- First step: PG load
  - The load of a single PG = *100 + (100 - read_ratio)*(pool_size - 1)*
  - For every 100 IOs on the system the primary performs 100 IOs and the secondaries perform only the write IOs
- Pool load = *PG load * PG num*
- Desired load per OSD = *pool load / osd num*
- If we can keep OSD load even across all OSDs the system will perform optimally (no weakest link in the chain)

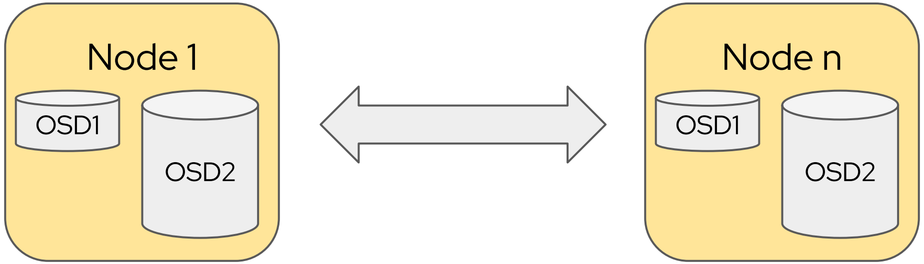
OSD Load Calculation

- The same for PG load calculator but we take part of PG for each OSD:
  - The load on a single OSD: *nPrims * 100 + (nPGs – nPrims) * write_ratio*
  - We want the OSD load to be close to X (calculated before) so we need to solve *nPrims * 100 + (nPGs – nPrims) * write_ratio = X* where the only variable is nPrims: *nPrims = (X – nPGs * write_ratio) / read_ratio*
    - When read_ratio is 0 there is no meaning for read balancing (and technically it means the parameter is not set)
  - **Note:** The load of a single OSD is in the range between *nPGs * 100* and *NPGs * write_ratio*

Red Hat

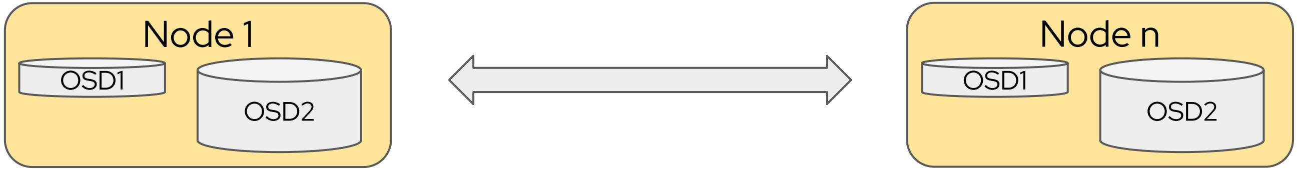
From math to accounting

- Now we need to move from theory to practice, let's look at the following configuration
  - Assume we want the small osd to be all primaries



From math to accounting

- Challenges
  - What happens if we have PG which maps to 2 (or 3) small OSDs
  - What happens if we have a PG which is mapped only to large OSDs
  - Here we need to move from math to accounting, which will give us worse performance than the math calculation
- The code implements iterative approach
  - First the all-primaries OSDs
  - Then no-primaries OSDs
  - Then math calculation



Results (calculation results from unit test)

```
>>>>>Desired primary distribution for read ratio: 70
osd.0: 16/6.35714 Load = current/desired 900/925
osd.1: 23/3.35714 Load = current/desired 1320/925
osd.2: 16/6.35714 Load = current/desired 480/925
osd.3: 12/8.07143 Load = current/desired 780/925
osd.4: 12/8.07143 Load = current/desired 780/925
osd.5: 12/8.07143 Load = current/desired 710/925
osd.6: 16/6.35714 Load = current/desired 1110/925
osd.7: 25/2.5 Load = current/desired 1030/925
osd.8: 11/8.5 Load = current/desired 680/925
osd.9: 16/6.35714 Load = current/desired 830/925
osd.10: 33/0 Load = current/desired 1620/990
<<<<<PGs distribution:
osd.0: 16/7 Load = 970
osd.1: 23/5 Load = 1040
osd.2: 16/0 Load = 480
osd.3: 12/9 Load = 990
osd.4: 12/7 Load = 850
osd.5: 12/9 Load = 990
osd.6: 16/7 Load = 970
osd.7: 25/4 Load = 1030
osd.8: 11/7 Load = 820
osd.9: 16/8 Load = 1040
osd.10: 33/1 Load = 1060
=== Read ratio: 70 High load before: 1620 High load after: 1060
 ====== end of iteration for read ratio 70
```

OSD weights

- Osd.0 - 50
- Osd.1 - 70
- Osd.2 - 50
- Osd.3 - 35
- Osd.4 - 35
- Osd.5 - 35
- Osd.6 - 50
- Osd.7 - 75
- Osd.8 - 35
- Osd.9 - 50
- Osd.10 - 100

Risks

- This feature is totally opt in, usage via `osdmaptool`
- No code on a critical path, minor changes for code on existing read balancer non critical path
- Based on `ceph osd pg-upmap-primary` which exists in Reef and is now in production
  - This command has a minor impact on the IO path, but it was Tech Preview in Reef and is part of Squid.
- Can be easily canceled (opt out)
- ⇒ Risk is very low

Limitations

- Works only on replicated pools
- Works only on pools with number of PGS is a power of 2 (all PGs have the same weight)
- Works only when all PGs are full
- For this version we do not honor OSD `primary_affinity`
  - If a user requires this, he should not use this feature
  - In future version we may honor OSD `primary_affinity 0` , but probably no other values (just 0 and 1)
- Works better from scratch than as an iterative process
  - This might be a bug - but it is not handled in Squid.
- In extreme cases may produce non optimal results

Red Hat

Thank you

Questions?

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat