Evaluating CephFS for Grid Storage

Lincoln Bryant • University of Chicago

OSG All Hands Meeting
Northwestern University
March 24, 2015

Today’s talk will be about this

- **CephFS** - POSIX filesystem built on top of the RADOS object store
  - Accessible via libcephfs + ‘ceph’ kernel module
  - Also via Ceph-FUSE

and not so much about these

- **RADOS** - Pure object store with global name space
  - Accessible via librados
- **RADOS Gateway** - S3/Swift APIs for the above
- RADOS Block Device (**RBD**) - Thinly provisioned block devices
  - Accessible via librbd + ‘rbd’ kernel module

Before we jump in...

- CephFS is not considered as ‘production ready’ as the other storage interfaces built ontop of RADOS
- CephFS requires at least kernel 3.10.x
    - If you’re using SL6, you’re stuck using Ceph-FUSE or rolling custom kernels.
- The Ceph metadata server (MDS) is currently not as robust as other components
    - Only a single active MDS, other MDS daemons on standby/failover
    - Multiple active MDS can be turned on, but here be dragons...

With that out of the way

- We've been using CephFS in production for a year and a half now with our Stash service.
- Users are consuming 250TB of Ceph storage
- Early growing pains but is generally reliable at this point.

Why CephFS over RBD, RADOSGW, etc?

- POSIX is a requirement
  - Our focus (OSG Connect) is on *user interaction.*
- CephFS can support GridFTP, XRootD, HTTP, SFTP, and protocol *du jour*.
  - Being POSIX makes this straightforward.
- RBD supports all of these too, but RBD can only be mounted writeable on one machine.
  - Re-export with NFS? :(:(:(

Erasure coding and cache tiering

A typical Ceph pool

- Consumes whole disks for object storage daemons (OSDs).
- Reliability comes from replication, not RAID on the underlying hardware
- Typical replication values are 2x or 3x.
  - At least half of raw storage is forfeit.

Intro to Erasure Coding

- Uses forward error correction (Hamming Codes) to provide generalized RAID-like functionality
- Implemented at the RADOS / pool level
- Each object is split into ***k* coding chunks** and ***m* parity chunks**
  - The cluster can lose *m* disks <u>or</u> *m* nodes without data loss.
  - *k*+*m* cannot exceed number of hosts in the cluster

Advantages of Erasure Coding

- In replicated configurations the amount of available storage is
  - **total storage / replica count**
  - Replication cuts down available disk quickly
- In erasure coded configurations, the amount of available storage is
  - **total storage * k/(k+m)**
  - BIG SAVINGS!
- All in all, you end up with more storage capacity and more redundancy with erasure coding.

However...

- Erasure coding is more CPU and network intensive.
  - In a replicated pool, objects are read from the primary copy
  - In an erasure coded pool, objects are reconstructed on the fly from *k* data chunks.
- CephFS doesn’t support erasure coding directly.
  - Erasure coded pools only support a subset of RADOS features, whereas CephFS uses almost all of them
  - We can work around this by placing a cache pool in front of the erasure coded pool

The Test Setup

Stash 2.0

- (14) Dell PowerEdge R730xd
  - (2) Intel Xeon E5-2650 v3 @ 2.30GHz
  - 96GB RAM
  - (12) 6TB, 7200 RPM SATA for Ceph
  - (2) 1TB SAS in RAID-1 for root
  - PERC H730 Mini w/ 1GB of onboard cache
- OS: Scientific Linux 6.6
- Ceph version: 0.93 (Hammer release candidate)
- Kernel version: 3.18.9 (Long-term support)
- All in all: **~1PB of raw storage.**

Ceph pool setup

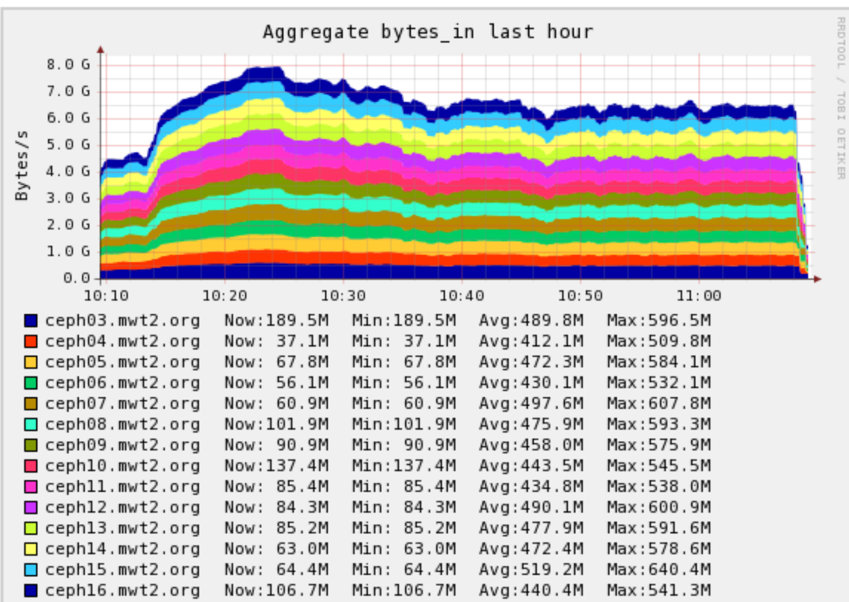
- *fs-data-ec* : Erasure coded pool, see next slide
- *fs-metadata*: Replicated pool (x3) for file attributes such as ownership, creation date, modification date, etc.
  - Upwards of 10-100GB, so don't bother erasure coding.
- *fs-hotpool* : Replicated (x2) cache tier pool for buffering writes to data pool
  - Flushes 'dirty' objects to EC pool at 40% fill
  - Evicts cached objects at 80% fill
  - Limited to 5TB (10TB after replication) of our storage

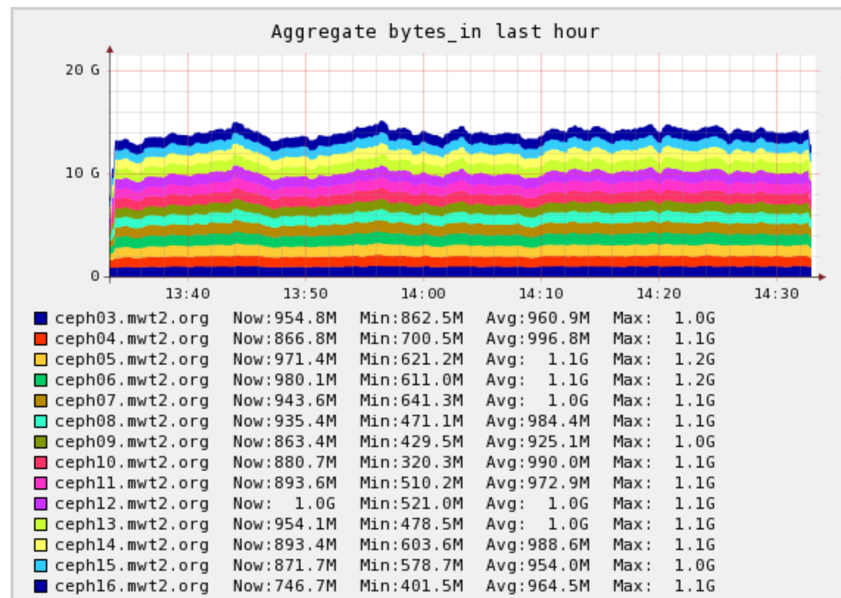
Erasure coding profile

- Ceph does not allow you to place multiple chunks of the same object on the same machine
- Since we have 14 machines, we decided to use the upper limit of ***k*+*m*=14**
- As for the actual values, we went with
  - **k = 11**
  - **m = 3**
- This lets us tolerate 3 simultaneously failed disks with no data loss
- We plan to re-integrate the old Stash (+100 disks) later, so early capacity planning is essential!

Raw pool performance

- On the left, clients writing into an erasure coded pool
- On the right, clients writing into a 2x replicated pool
- Erasure coding at least 40% slower





CephFS - Writes

throughput	single 10Gb client	multiple 10Gb clients (5)
single instance of ‘dd’	**~175 MB/s**	~814 MB/s
multiple instances of ‘dd’	~471 MB/s	**~1885 MB/s**

- Client tool: *dd* with *conv=fdatasync* option
- 4GB file, *tmpfs* to CephFS copy
- Single thread, single client copy is important for end-users.
  - No one is going to realistically split their ‘cp’ into multiple runs
- Multi-thread, multi-client is what we’ll realistically see for an SE.

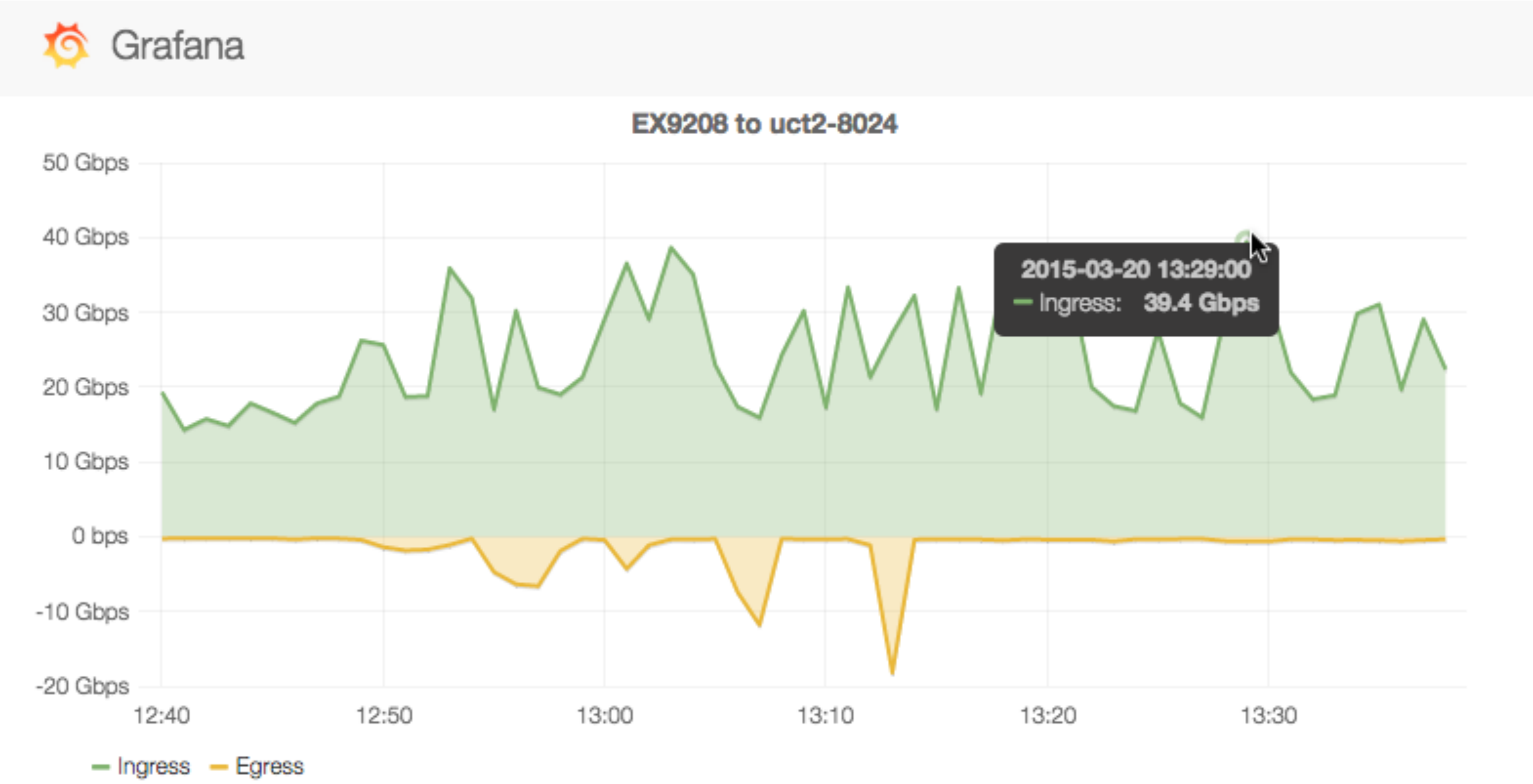
CephFS - Reads

throughput	single 10Gb client	multiple 10Gb clients (5)
single instance of ‘dd’	**~193 MB/s**	~854 MB/s
multiple instances of ‘dd’	~853 MB/s	**~3720 MB/s**

- Again using ‘dd’ to copy a 4GB file out of CephFS into /dev/null
- These reads fit entirely in the cache pool, so the performance is more like a traditional replicated setup.

CephFS - Reads

- Possibly performance could go higher, my network plots indicate I’ve maxed out my 40Gb bond..



SRM/GridFTP

- Stood up an SRM instance with BeStMan
  - From start to finish, basic setup took me 2-3 hours.
  - Kudos to the BeStMan team & OSG documentation for making this straight-forward
- Copying files from OSG Connect login node to *ceph-se.osgconnect.net* SRM/gridFTP and back:
  - ~328 MB/s writes from memory
  - ~386 MB/s reads to /dev/null
- Somewhat comparable to 'dd' results.
  - GridFTP door is a VM on a shared hypervisor

Testing summary

- Single threaded performance is very acceptable for user interaction
- Erasure coding is at least **40% slower** than replication at the pool level, but cache tiering helps significantly
- With a k=11,m=3 configuration and cache tiering, we get **75-78%** of our raw storage **usable**
- Based on the ‘dd’ results, we could stand up 4-5 GridFTP doors to get the most out of Ceph
- Since the cache tier is replicated, need to be careful with sizing & when to flush dirty objects

Thoughts and recommendations on operations

Huge nodes can be tricky

- Stash v1.0 was much like our standard dCache purchase:
  - Each box has several controllers and a boatload of attached storage (something like 60 disks per machine)
  - Definitely bottlenecked at the single 10Gbps interface
  - RAM has never really been a problem, but CPUs were definitely overloaded during recovery ops.
  - Good rule of thumb: One machine should be no more than 10% of your total storage.

Our growing pains

- With so many disks, we ran into file descriptor and process limits.
    - I think we’ve all seen this at least once in this line of work
- Created the cluster with too few placement groups.
    - Easily remedied, but increasing the placement groups in an active pool triggers massive data redistribution.

Bugs, deficiencies, etc

- We got burned on kernels a few times
  - In earlier kernels (3.8ish?), files in CephFS would disappear then resurface when stat'd
  - Bugs in the ACL code before 3.18 can cause a lot of "????" in 'ls'. Remedied with 'noacl' mount option
- In Ceph v0.72, the metadata server (MDS) would not become 'active' unless the cluster was HEALTH_OK
  - This creates a hung mount and user complaints if the MDS goes down during recovery
- OSD and CephFS kernel mounts on the same machine generally not recommended
  - Potential deadlock issues in certain scenarios

Other things of note

- CephFS does not yet have ‘fsck’. This is one of the primary reasons it’s not considered “production ready” today.
- CephFS and NFS are both in-kernel clients of the TCP/IP stack. Exporting CephFS as NFS can lead to network queue prioritization conflicts, thus deadlocks. Solutions:
  - CephFS capability in updated kernel (don’t need NFS)
  - Ceph-FUSE (move CephFS into userspace)
  - Ganesha NFS (move NFS into userspace)
- Ceph-FUSE tends to lag behind the kernel client
- In some corner cases, CephFS differs from POSIX
  - http://ceph.com/docs/giant/dev/differences-from-posix/

Open questions

- What happens in the scenario where the cache is full, and data is writing into the cache faster than the cache can backfill the EC pool?
  - Does the filesystem claim its full?
  - Do transfers get throttled?
  - Or does something nasty happen?
- Does it make sense to use FSCache for clients?
  - Introduced in Kernel 3.12 - 3.13
    - CONFIG_CEPH_FSCACHE=y
  - Certainly will improve latency, but what about throughput?

Questions?