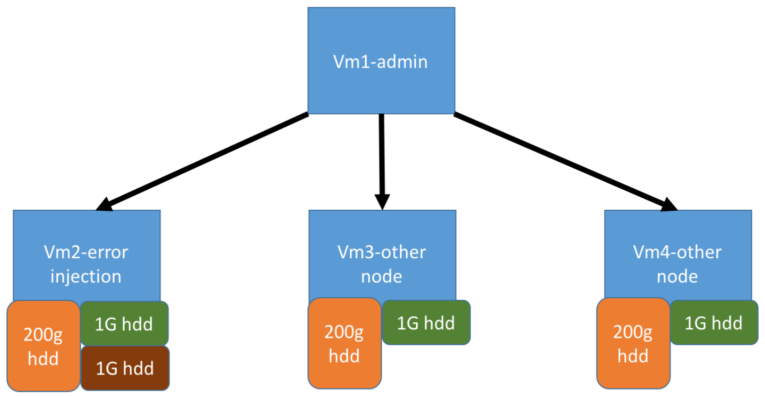
Distributed System Project 2

Yuan-Ting Hsieh yhsieh28@wisc.edu
Hsuan-Heng Wu hwu337@wisc.edu

In this project, we are trying to use errfs from CORDS to test the robustness of Ceph storage.

1. Setup: We install Ceph 13.2.2 on 3 nodes of Ubuntu 18.04 with the underlying local storage engine filestore. We create a pool with 100 placement groups, data size 3 and data min_size 1 or data min_size 3. (project1_install.h)
2. Workflow of using errfs to get the trace or do cords (trace.py, cords.py):
    a. Modify the config in /etc/ceph/ceph.conf, set OSD data to the mount point of errfs (addconfig.py)
    b. Mount errfs
    c. Stop OSD
    d. Start OSD
    e. Run workload
    f. Stop OSD
    g. Unmount errfs
3. Workload:
    a. Setup: Put a text file contains number 0~200 into the pool five times using "rados put -p [pool] testObj[i] test.txt"
    b. Try to get the object from Ceph using "rados get -p [pool] testObj[i] newtest.txt"

Setup:



(Backup journal)

Findings:

1. Ceph:
    a. If we put a large file into Ceph without providing the --striper flag, the attempt will fail but garbage will remain in Ceph
    b. Can use "sudo ceph osd metadata $ID | grep -e id -e hostname -e osd_objectstore" to check underlying storage
2. Errfs:
    a. Fault Injection:
        i. read: corrupt garbage/junk(cg), corrupt zero(cz), error io(eio)
        ii. write: error io(eio)

        iii. append: error io(eio), not enough space(esp)
    b. If we just touch a file, errfs won't write anything to its trace file
3. Run errfs with Ceph:
    a. Ownership is the key:
        i. What user are you using to call errfs? => pass allow_other flag
        ii. What user are you using to start Ceph? => use foreground mode to set the user to Ceph
        iii. Who is the owner of trace files? => we use chmod 777 for trace files
        iv. Who is the owner of each of the files in the /var/lib/ceph/osd/ceph-x directory? => use chown
    b. What part of Ceph traces could we inject errors?
        i. keyring, type, magic, whoami, ceph_fsid, fsid, store_version, superblock, fiemap_test, meta, these are all metadata, injecting errors on these would cause the ceph osd to crash so we do not inject errors on these traces.
        ii. We mainly inject errors in traces relate to omap and Ceph objects file
    c. When running cords, at first, we frequently start and stop the Ceph OSD system service using systemctl. Which will cause “Start request repeated too quickly. Failed with result 'start-limit-hit'” error. The solutions we found suggest modify “StartLimitIntervalSec” and “StartLimitBurst” in servicemd's configuration, but that does not work for us. We instead doing “systemctl daemon-reload” for every 20 runs because the quota is 30 runs. By doing reload we are resetting the counter.
    d. But using systemctl to start osd is really cumbersome because of the restart behavior. So we change to use foreground mode (sudo osd ceph -f)
    e. Previously we didn’t pass -x flag when creating snapshots, then Ceph cannot find XAttrs and could not successfully launch, we’ve added -x flag to fix that
    f. Previously we didn’t comment out assert(false) in err_fs to allow symlink

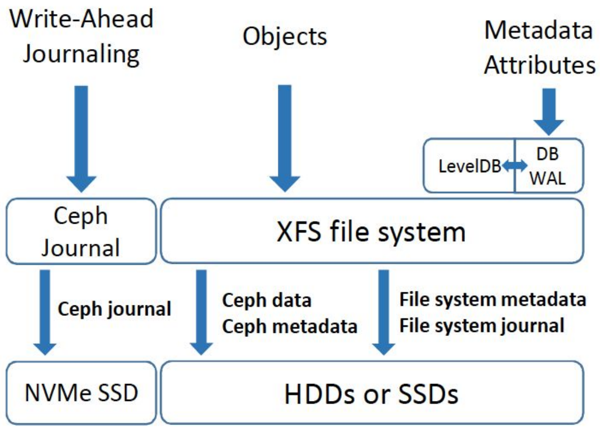
4. Analysis:
    a. Steps in Ceph to restart OSD:
        i. Get file storage structures from fiemap
        ii. When XAttr is missing, start omap initialization using RocksDB
            1. Detect MANIFEST, sst and log file
            2. Try to recover from MANIFEST
            3. Try to recover logs
            4. Try to append to sst
        iii. If XAttr is present, Ceph will abort when it is trying to fetch epoch value for OSD map because it gets zero bytes, which cause the function to return an uninitialized object with potentially zero epoch value that triggers an assertion failure in the check_new_interval call. Ceph will keep retrying until we manually kill it.

    b. Out first observation when XAttr is missing: When running our read task, there are some files related to omap to inject error, 000012.sst and 000010.log (behavior handled by RocksDB)
        i. If inject errors in the log, then try to recover from logs would fail, and RocksDB will revert the log back to a certain point and try to write changes to the new version of MANIFEST

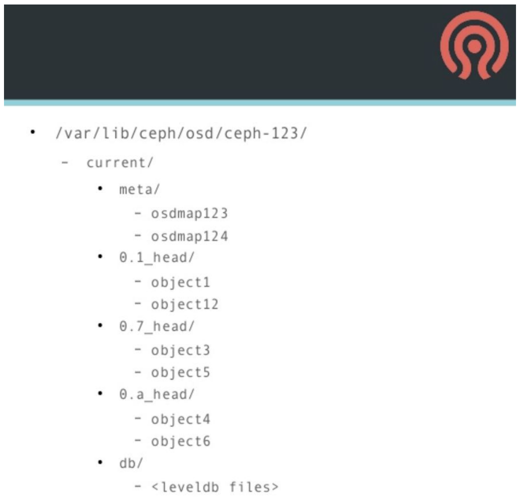
    ii. If inject errors in sst, then we can successfully recover log. But when it tries to append to sst, it would fail
    iii. Either case the osd init would fail, but the overall system can function

c. After we fix those issues, we could also inject errors on the trace of Ceph objects (ex./var/lib/ceph/osd/ceph-0/current/1.2d_head/testObj0__head_EE68AEED__1)
    i. After injection, if our osd pool min size is set to 1 then we can still get the correct object
    ii. After injection, if our osd pool min size is set to 3
        1. In the case of EIO, Ceph report getting an unexpected error when doing the transaction, and will hang forever after aborting several threads.
        2. Inject cg/cz: Ceph will detect that the crc checksum doesn't match and try to get copies from other two osds and provide the correct result.

d. We refer to the below slides to understand the structure of the OSD files with FileStore backend, something different from these slides is that Ceph switch from LevelDB to RocksDB and that it stores metadata under omap/ not db/



(a) FileStore



- /var/lib/ceph/osd/ceph-123/
  - current/
    - meta/
      - osdmap123
      - osdmap124
    - 0.1_head/
      - object1
      - object12
    - 0.7_head/
      - object3
      - object5
    - 0.a_head/
      - object4
      - object6
    - db/
      - <leveldb files>

References:

[1] http://research.cs.wisc.edu/adsl/Publications/cords-tos17.pdf
[2] http://storageconference.us/2017/Papers/CephObjectStore.pdf
[3]https://www.slideshare.net/sageweil1?utm_campaign=profiletracking&utm_medium=sssite&utm_source=ssslideview
[4] https://ceph.com/geen-categorie/ceph-osd-where-is-my-data/