CephFS Development Update

John Spray
john.spray@redhat.com

Vault 2015

redhat.

Agenda

- Introduction to CephFS architecture
- Architectural overview
- What's new in Hammer?
- Test & QA

Distributed filesystems are hard

Object stores scale out well

- Last writer wins consistency
- Consistency rules only apply to one object at a time
- Clients are stateless (unless explicitly doing lock ops)
- No relationships exist between objects
- Scale-out accomplished by mapping objects to nodes
- Single objects may be lost without affecting others

POSIX filesystems are hard to scale out

- Extents written from multiple clients must win or lose on all-or-nothing basis → locking
- Inodes depend on one another (directory hierarchy)
- Clients are stateful: holding files open
- Scale-out requires spanning inode/dentry relationships across servers
- Loss of data can damage whole subtrees

Failure cases increase complexity further

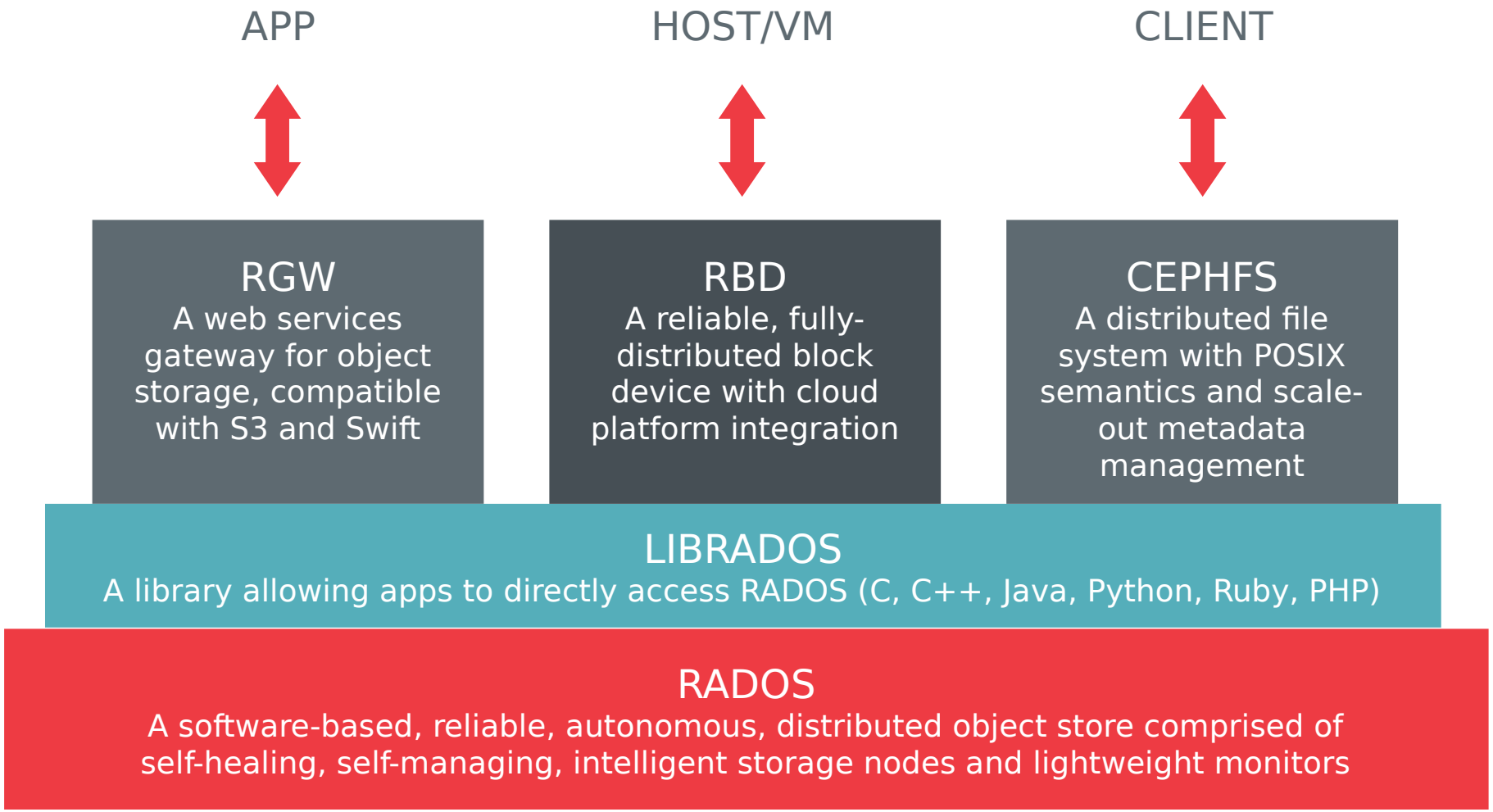
- What should we do when... ?
  - Filesystem is full
  - Client goes dark
  - Server goes dark
  - Memory is running low
  - Clients misbehave
- Hard problems in distributed systems generally, especially hard when we have to uphold POSIX semantics designed for local systems.

So why bother?

- Because it's an interesting problem :-)
- Filesystem-based applications aren't going away
- POSIX is a lingua-franca
- Containers are more interested in file than block

Architectural overview

Ceph architecture

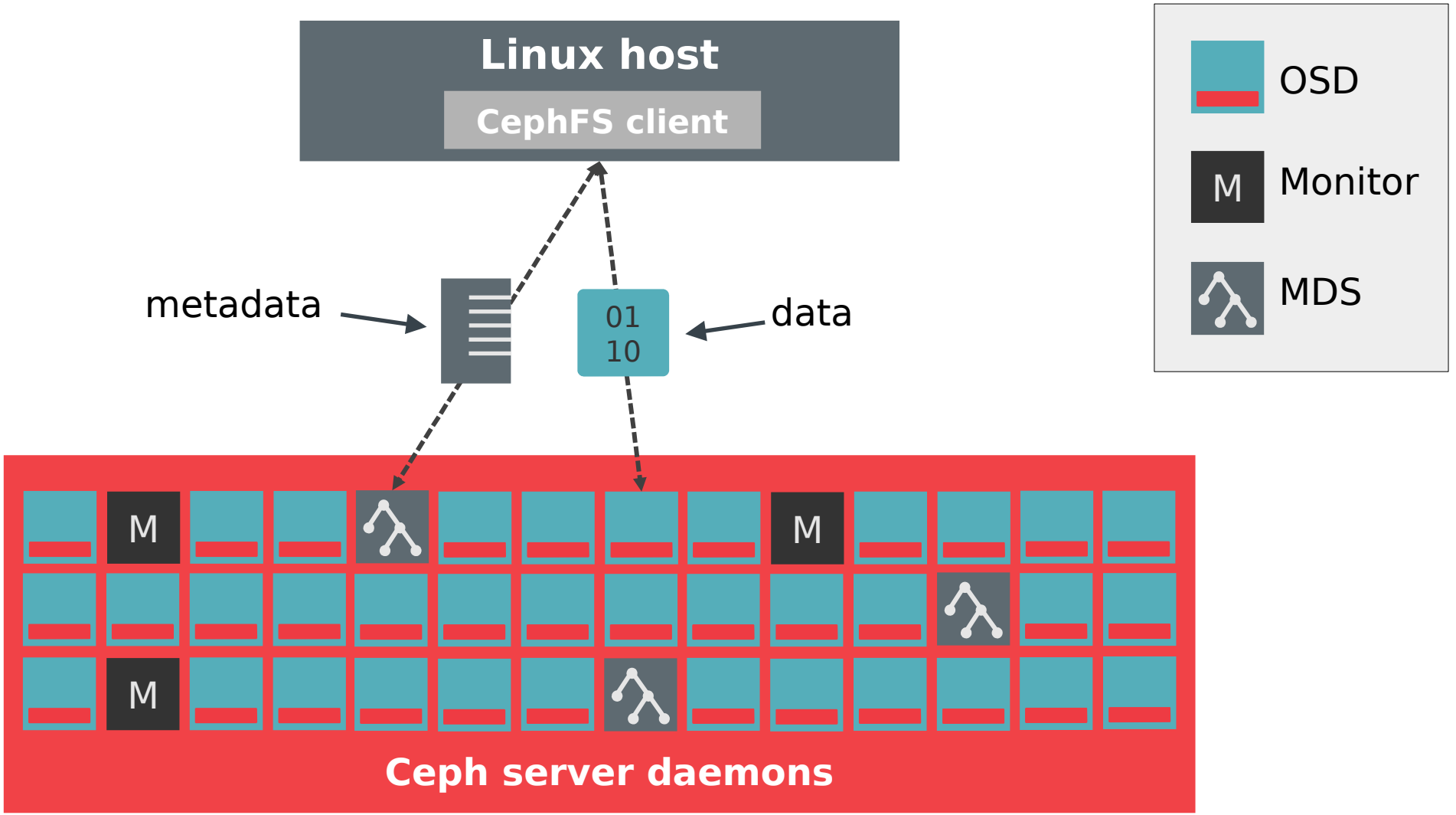


CephFS architecture

- Inherit resilience and scalability of RADOS
- Multiple metadata daemons (MDS) handling dynamically sharded metadata
- Fuse & kernel clients: POSIX compatibility
- Extra features: Subtree snapshots, recursive statistics

Weil, Sage A., et al. "Ceph: A scalable, high-performance distributed file system." Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, 2006.
http://ceph.com/papers/weil-ceph-osdi06.pdf

Components


Linux host
CephFS client
metadata
01
10
data
OSD
M
Monitor
MDS
M
M
M
Ceph server daemons

Use of RADOS for file data

- File data written directly from clients
- File contents striped across RADOS objects, named after <inode>.<offset>

```
# ls -i myfile
1099511627776 myfile
# rados -p cephfs_data ls
10000000000.00000000
10000000000.00000001
```

- Layout includes which pool to use (can use diff. pool for diff. directory)
- Clients can modify layouts using ceph.* vxattrs

Use of RADOS for metadata

- Directories are broken into fragments
- Fragments are RADOS OMAPs (key-val stores)
- Filenames are the keys, dentries are the values
- Inodes are embedded in dentries

- Additionally: inode *backtrace* stored as xattr of first data object. Enables direct resolution of hardlinks.

RADOS objects: simple example

```
# mkdir mydir ; dd if=/dev/urandom bs=4M count=3 mydir/myfile1
```

Metadata pool

1.00000000	
mydir1	*10000000001*

10000000001.00000000	
myfile1	10000000002

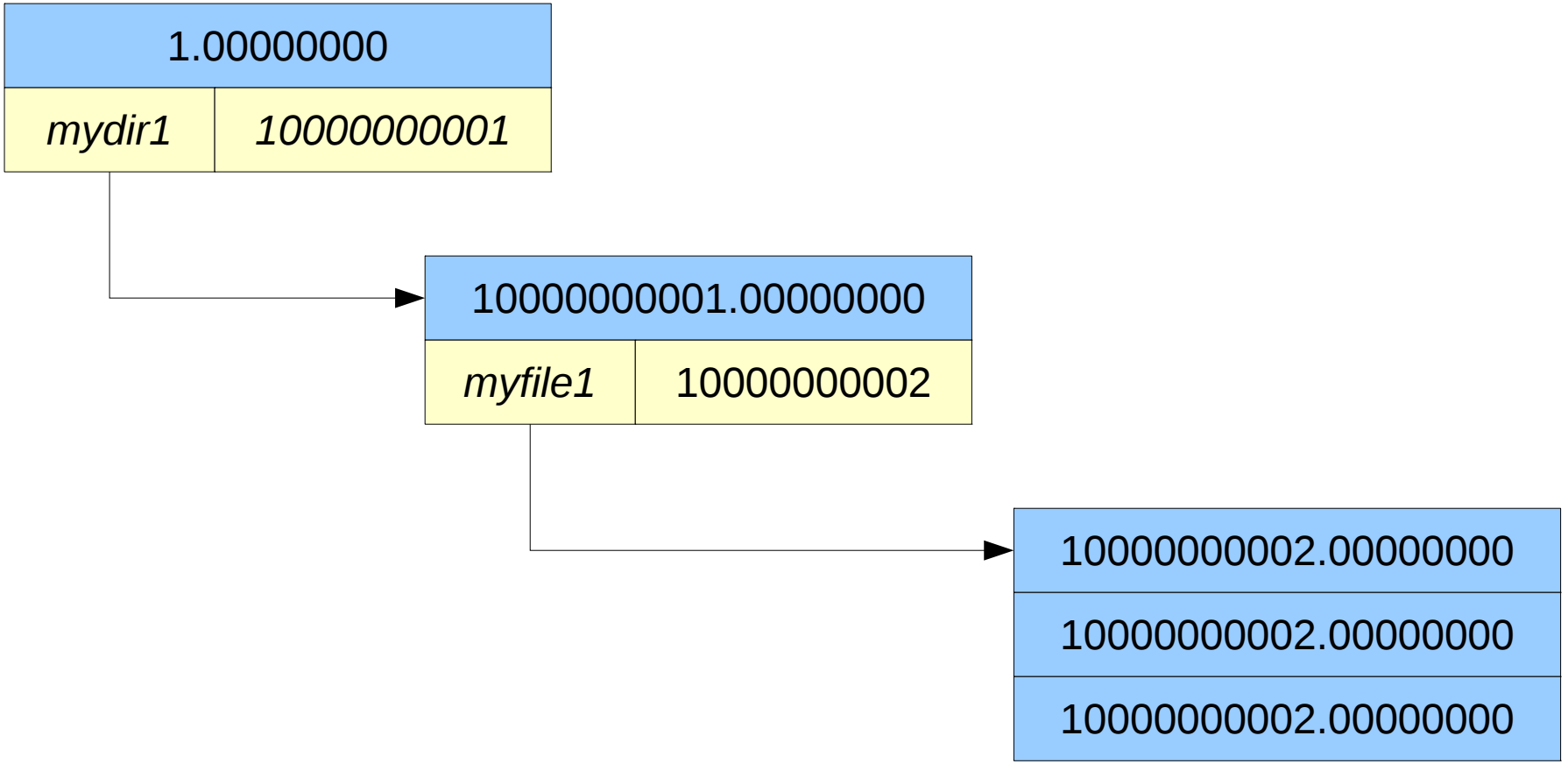
Data pool

10000000002.00000000	
parent	*/mydir/myfile1*

10000000002.00000001

10000000002.00000002

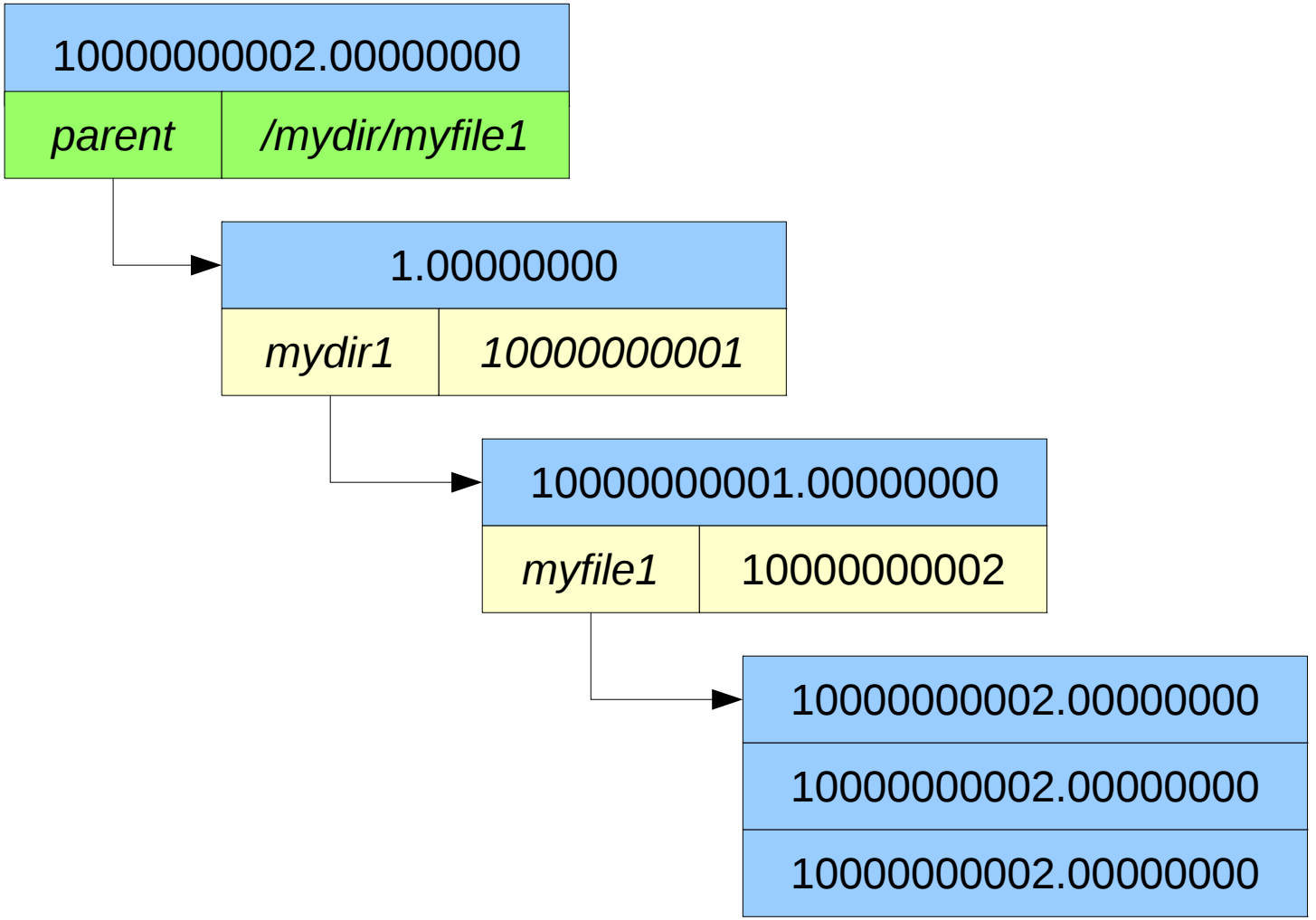
Normal case: lookup by path



Lookup by inode

- Sometimes we need inode → path mapping:
  - Hard links
  - NFS handles
- Costly to store this: mitigate by piggybacking paths (*backtraces*) onto data objects
  - Con: storing metadata to data pool
  - Con: extra IOs to set backtraces
  - Pro: disaster recovery from data pool

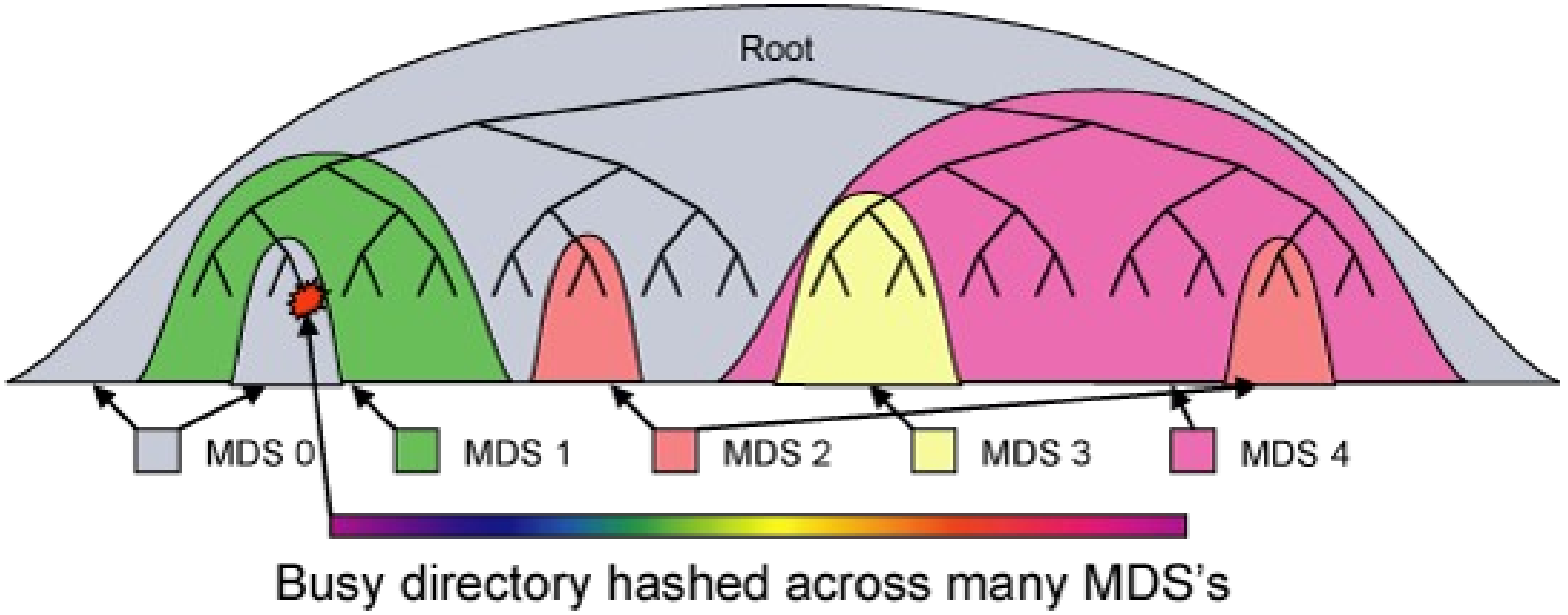
Lookup by inode



The MDS

- MDS daemons do nothing (standby) until assigned an identity (*rank*) by the RADOS monitors (active).
- Each MDS rank acts as the authoritative cache of some subtrees of the metadata on disk
- MDS ranks have their own data structures in RADOS (e.g. journal)
- MDSs track usage statistics and periodically globally renegotiate distribution of subtrees
- ~63k LOC

Dynamic subtree placement



Client-MDS protocol

- Two implementations: ceph-fuse, kclient
- Client learns MDS addrs from mons, opens session with each MDS as necessary
- Client maintains a cache, enabled by fine-grained *capabilities* issued from MDS.
- On MDS failure:
  - *reconnect* informing MDS of items held in client cache
  - *replay* of any metadata operations not yet known to be persistent.
- Clients are fully trusted (for now)

Detecting failures

- MDS:
  - “beacon” pings to RADOS mons. Logic on mons decides when to mark an MDS failed and promote another daemon to take its place
- Clients:
  - “RenewCaps” pings to each MDS with which it has a session. MDSs individually decide to drop a client's session (and release capabilities) if it is too late.

CephFS in practice

```
ceph-deploy mds create myserver
ceph osd pool create fs_data
ceph osd pool create fs_metadata
ceph fs new myfs fs_metadata fs_data

mount -t cephfs x.x.x.x:6789 /mnt/ceph
```

Development update

Towards a production-ready CephFS

- Focus on resilience:
  - Handle errors gracefully
  - Detect and report issues
  - Provide recovery tools
- Achieve this first within a conservative single-MDS configuration
- ...and do lots of testing

Statistics in Firefly->Hammer period

- Code:
  - src/mds: 366 commits, 19417 lines added or removed
  - src/client: 131 commits, 4289 lines
  - src/tools/cephfs: 41 commits, 4179 lines
  - ceph-qa-suite: 4842 added lines of FS-related python
- Issues:
  - 108 FS bug tickets resolved since Firefly (of which 97 created since firefly)
  - 83 bugs currently open for filesystem, of which 35 created since firefly
  - 31 feature tickets resolved

New setup steps

- CephFS data/metadata pools no longer created by default
- CephFS disabled by default
- New `fs [new|rm|ls]` commands:
  - Interface for potential multi-filesystem support in future
- Setup still just a few simple commands, while avoiding confusion from having CephFS pools where they are not wanted.

MDS admin socket commands

- `session ls`: list client sessions
- `session evict`: forcibly tear down client session
- `scrub_path`: invoke scrub on particular tree
- `flush_path`: flush a tree from journal to backing store
- `flush journal`: flush everything from the journal
- `force_readonly`: put MDS into readonly mode
- `osdmap barrier`: block caps until this OSD map

MDS health checks

- Detected on MDS, reported via mon
  - Client failing to respond to cache pressure
  - Client failing to release caps
  - Journal trim held up
  - ...more in future
- Mainly providing faster resolution of client-related issues that can otherwise stall metadata progress
- Aggregate alerts for many clients
- Future: aggregate alerts for one client across many MDSs

OpTracker in MDS

- Provide visibility of ongoing requests, as OSD does

```
ceph daemon mds.a dump_ops_in_flight
{
    "ops": [
        {
            "description": "client_request(client.
            "initiated_at": "2015-03-10 22:26:17.4
            "age": 0.052026,
            "duration": 0.001098,
            "type_data": [
                "submit entry: journal_and_reply",
                "client.4119:21120",
...
```

FSCK and repair

- Recover from damage:
  - Loss of data objects (which files are damaged?)
  - Loss of metadata objects (what subtree is damaged?)
- Continuous verification:
  - Are recursive stats consistent?
  - Does metadata on disk match cache?
  - Does file size metadata match data on disk?

Learn more in *CephFS fsck: Distributed File System Checking* - Gregory Farnum, Red Hat (Weds 15:00)

cephfs-journal-tool

- Disaster recovery for damaged journals:
  - inspect/import/export/reset
  - header get/set
  - event recover_dentries
- Works in parallel with new journal format, to make a journal glitch non-fatal (able to skip damaged regions)
- Allows rebuild of metadata that exists in journal but is lost on disk
- Companion **cephfs-table-tool** exists for resetting session/inode/snap tables as needed afterwards.

Full space handling

- Previously: a full (95%) RADOS cluster stalled clients writing, but allowed MDS (metadata) writes:
  - Lots of metadata writes could continue to 100% fill cluster
  - Deletions could deadlock if clients had dirty data flushes that stalled on deleting files
- Now: generate ENOSPC errors in the client, propagate into fclose/fsync as necessary. Filter ops on MDS to allow deletions but not other modifications.
- Bonus: I/O errors seen by client also propagated to fclose/fsync where previously weren't.

OSD epoch barrier

- Needed when synthesizing ENOSPC: ops in flight to full OSDs can't be cancelled, so must ensure any subsequent I/O to file waits for later OSD map.
- Same mechanism needed for client eviction: once evicted client is blacklisted, must ensure other clients don't use caps until version of map with blacklist has propagated.
- Logically this is a per-file constraint, but much simpler to apply globally, and still efficient because:
  - Above scenarios are infrequent
  - On a healthy system, maps are typically propagated faster than our barrier

Client management

- Client metadata
  - Reported at startup to MDS
  - Human or machine readable
- Stricter client eviction
  - For misbehaving, not just dead clients

Client management: metadata

```
# ceph daemon mds.a session ls
...
  "client_metadata": {
    "ceph_sha1": "a19f92cf...",
    "ceph_version": "ceph version 0.93...",
    "entity_id": "admin",
    "hostname": "claystone",
    "mount_point": "\/home\/john\/mnt"
  }
```

- Metadata used to refer to clients by hostname in health messages
- Future: extend to environment specific identifiers like HPC jobs, VMs, containers...

Client management: strict eviction

```
ceph osd blacklist add <client addr>

ceph daemon mds.<id> session evict

ceph daemon mds.<id> osdmap barrier
```

- Blacklisting clients from OSDs may be overkill in some cases if we know they are already really dead.
- This is fiddly when multiple MDSs in use: should wrap into a single global evict operation in future.
- Still have timeout-based non-strict (MDS-only) client eviction, in which clients may rejoin. Potentially unsafe: new mechanism may be needed.

FUSE client improvements

- Various fixes to cache trimming
- FUSE issues since linux 3.18: lack of explicit means to dirty cached dentries en masse (we need a better way than remounting!)
- `flock` is now implemented (require fuse >= 2.9 because of interruptible operations)
- Soft client-side quotas (stricter quota enforcement needs more infrastructure)

Test, QA, bug fixes

- The answer to “Is CephFS ready?”
- *teuthology* test framework:
  - Long running/thrashing test
  - Third party FS correctness tests
  - Python functional tests
- We dogfood CephFS within the Ceph team
  - Various kclient fixes discovered
  - Motivation for new health monitoring metrics
- **Third party testing is extremely valuable**

Functional testing

- Historic tests are “black box” client workloads: no validation of internal state.
- More invasive tests for exact behaviour, e.g.:
  - Were RADOS objects really deleted after a *rm*?
  - Does MDS wait for client reconnect after restart?
  - Is a hardlinked inode relocated after an unlink?
  - Are stats properly auto-repaired on errors?
  - Rebuilding FS offline after disaster scenarios
- Fairly easy to write using the classes provided:

```
ceph-qa-suite/tasks/cephfs
```

Future

- Priority: Complete FSCK & repair tools
- Other work:
  - Multi-MDS hardening
  - Snapshot hardening
  - Finer client access control
  - Cloud/container integration (e.g. Manilla)

Tips for early adopters

http://ceph.com/resources/mailing-list-irc/

http://tracker.ceph.com/projects/ceph/issues

http://ceph.com/docs/master/rados/troubleshooting/log-and-debug/

- Does the most recent development release or kernel fix your issue?
- What is your configuration? MDS config, Ceph version, client version, kclient or fuse
- What is your workload?
- Can you reproduce with debug logging enabled?

Questions?

CephFS user tips

- Choose MDS servers with lots of RAM
- Investigate clients when diagnosing stuck/slow access
- Use recent Ceph and recent kernel
- Use a conservative configuration:
  - Single active MDS, plus one standby
  - Dedicated MDS server
  - A recent client kernel, or the fuse client
  - No snapshots, no inline data
- Test it aggressively: especially through failures of both clients and servers.

Journaling and caching in MDS

- Metadata ops initially written ahead to MDS journal (in RADOS).
  - I/O latency on metadata ops is sum of network latency and journal commit latency.
  - Metadata remains pinned in in-memory cache until expired from journal.
- Keep a long journal: replaying the journal after a crash warms up the cache.
- Control cache size with mds_cache_size. Trimming oversized caches is challenging, because relies on cooperation from clients and peer MDSs. Currently simple LRU.

More perf counters

```
$ ceph daemonperf mds.a
```

-----mds------			--mds_server--			---objecter---			-----mds_cache-----				---mds_log----		
rlat	inos	caps	hsr	hcs	hcr	writ	read	actv	recd	recy	stry	purg	segs	evts	subm
3	757	98	0	0	106	18	0	0	0	0	649	0	5	3.6k	212
8	821	162	0	0	53	655	0	640	0	0	643	6	5	2.7k	130
6	821	162	0	0	81	12	0	545	0	0	536	107	5	3.1k	376
6	821	162	0	0	50	5	0	270	0	0	264	272	5	3.8k	642
6	821	162	0	0	54	10	0	38	0	0	22	242	6	4.3k	594
5	877	401	0	0	104	5	0	0	0	0	22	0	6	4.6k	228
8	532	500	0	0	57	349	0	294	0	0	11	11	4	2.8k	228
6	594	561	0	0	54	8	0	123	0	0	11	0	4	2.9k	154
14	629	597	0	0	78	17	0	0	0	0	11	0	4	3.0k	136
5	711	679	0	1	84	15	0	0	0	0	11	0	5	3.3k	304
4	750	717	0	0	62	3	0	0	0	0	11	0	5	3.5k	202
4	823	791	0	0	81	131	0	98	0	0	7	4	5	2.6k	142
6	862	830	0	0	45	8	0	0	0	0	7	0	4	2.7k	108
5	862	830	0	0	46	0	0	0	0	0	7	0	4	2.8k	134
6	862	830	0	0	45	0	0	0	0	0	7	0	4	3.0k	188
6	862	830	0	0	48	0	0	0	0	0	7	0	4	3.2k	180
17	1.0k	1.0k	0	0	33	373	0	326	0	0	7	0	4	2.4k	186
18	1.0k	1.0k	0	0	37	10	0	160	0	0	7	0	4	2.5k	146
12	1.0k	1.0k	0	0	43	4	0	0	0	0	7	0	4	2.7k	192
5	1.0k	1.0k	0	0	47	0	0	0	0	0	7	0	4	2.9k	194
4	1.2k	1.1k	0	0	60	2	0	0	0	0	7	0	4	3.1k	174
13	1.2k	1.2k	0	0	27	272	0	209	0	0	0	7	4	2.3k	194