ceph: distributed storage for cloud infrastructure

sage weil
msst – april 16, 2012

outline

- motivation
- overview
- how it works
  - architecture
  - data distribution
  - rados
  - rbd
  - distributed file system
- practical guide, demo
  - hardware
  - installation
  - failure and recovery
  - rbd
  - libvirt
- project status

storage requirements

- scale
  - terabytes, petabytes, exabytes
  - heterogeneous hardware
  - reliability and fault tolerance
- diverse storage needs
  - object storage
  - block devices
  - shared file system (POSIX, coherent caches)
  - structured data

time

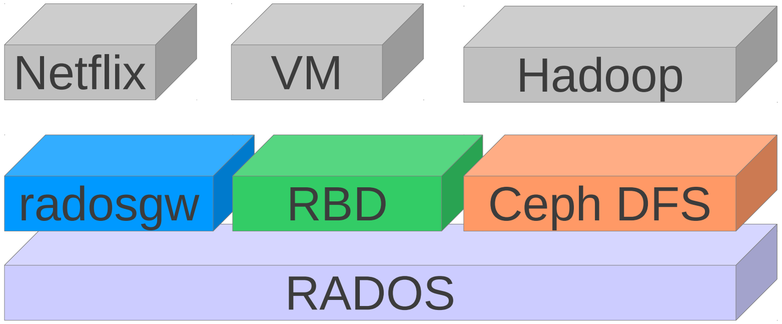
- ease of administration
- no manual data migration, load balancing
- painless scaling
  - expansion **and** contraction
  - seamless migration

money

- low cost per gigabyte
- no vendor lock-in
- software solution
- commodity hardware
- open source

ceph: unified storage system

- objects
  - small or large
  - multi-protocol
- block devices
  - snapshots, cloning
- files
  - cache coherent
  - snapshots
  - usage accounting



Netflix

VM

Hadoop

radosgw

RBD

Ceph DFS

RADOS

open source

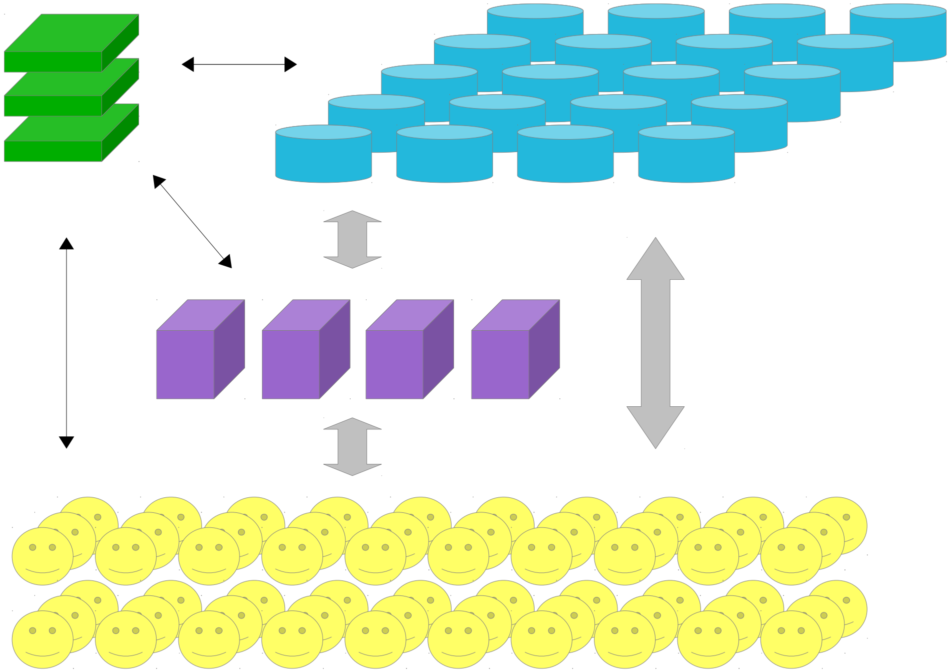
- LGPLv2
  - copyleft
  - free to link to proprietary code
- no copyright assignment
  - no dual licensing
  - no “enterprise-only” feature set

distributed storage system

- data center (not geo) scale
  - 10s to 10,000s of machines
  - terabytes to exabytes
- fault tolerant
  - no SPoF
  - commodity hardware
    - ethernet, SATA/SAS, HDD/SSD
    - RAID, SAN probably a waste of time, power, and money

architecture

- monitors (ceph-mon)
  - 1s-10s, paxos
  - lightweight process
  - authentication, cluster membership, critical cluster state
- object storage daemons (ceph-osd)
  - 1s-10,000s
  - smart, coordinate with peers
- clients (librados, librbd)
  - zillions
  - authenticate with monitors, talk directly to ceph-osds
- metadata servers (ceph-mds)
  - 1s-10s
  - build POSIX file system on top of objects



rados object storage model

- pools
  - 1s to 100s
  - independent namespaces or object collections
  - replication level, placement policy
- objects
  - trillions
  - blob of data (bytes to gigabytes)
  - attributes (e.g., “version=12”; bytes to kilobytes)
  - key/value bundle (bytes to gigabytes)

object storage daemons

- client/server, host/device paradigm doesn't scale
  - idle servers are wasted servers
  - if storage devices don't coordinate, clients must
- ceph-osds are intelligent storage daemons
  - coordinate with peers
  - sensible, cluster-aware protocols
- flexible deployment
  - one per disk
  - one per host
  - one per RAID volume
- sit on local file system
  - btrfs, xfs, ext4, etc.

data distribution

- all objects are replicated N times
- objects are automatically placed, balanced, migrated in a dynamic cluster
- must consider physical infrastructure
  - ceph-osds on hosts in racks in rows in data centers

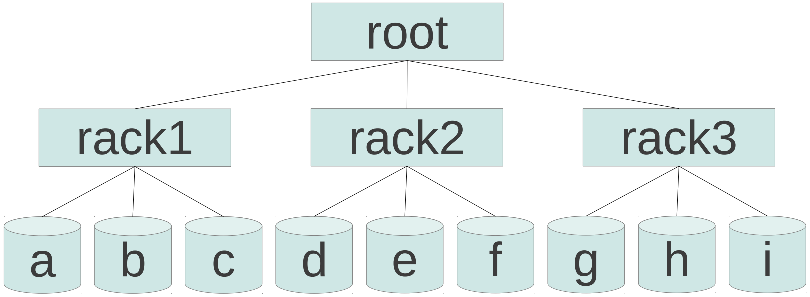
- three approaches
  - pick a spot; remember where you put it
  - pick a spot; write down where you put it
  - calculate where to put it, where to find it

CRUSH

- pseudo-random placement algorithm
  - uniform, weighted distribution
  - fast calculation, no lookup
- hierarchial
  - tree reflects physical infrastructure
- placement rules
  - “3 replicas, same row, different racks”
- stable: predictable, bounded migration on changes
  - N → N + 1 ceph-osds means a bit over 1/Nth of data moves

placement process

- device hierarchy reflects infrastructure
- *choose* function
  - sample pseudorandom decent
  - sequence of possible choices
  - return first N unique and acceptable values
  - parameterized by
    - x (id/hash of object)
    - tree (node ids, types, weights)
    - device state (in/out)



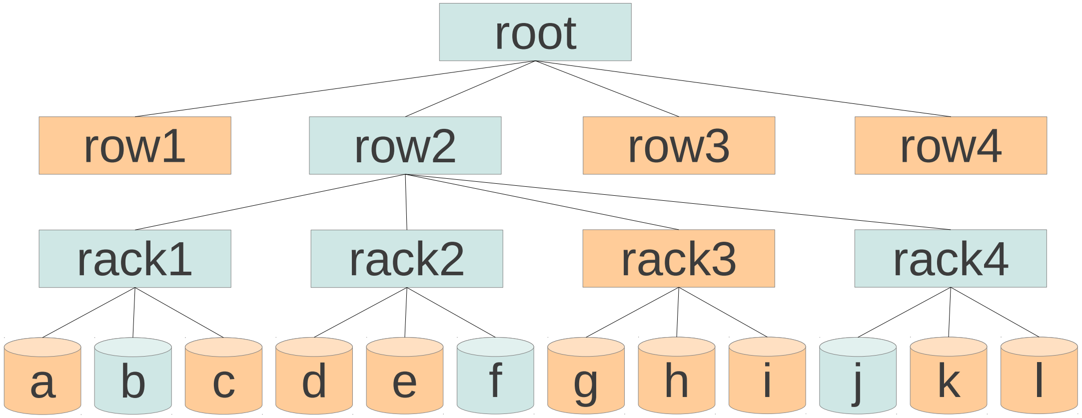
- b, e, d, g, ~~b~~, i, ~~e~~, h, ...

→ b, e, d

…

placement process (2)

- rules rely on node types
- example
  - take(root)
  - choose(1, row)
  - choose(3, rack)
  - choose(1, device)
  - emit
- or
  - take(fast)
  - choose(1, device)
  - emit
  - take(slow)
  - choose(2, rack)
  - choose(1, device)
  - emit



- root
- row2
- rack2, rack1, rack4
- f, b, j

…

placement and data safety

- separate replicas across failure domains
  - power circuits
  - switches, routers
  - physical location
- important for declustered replication
  - replicas from one device are spread across many other devices

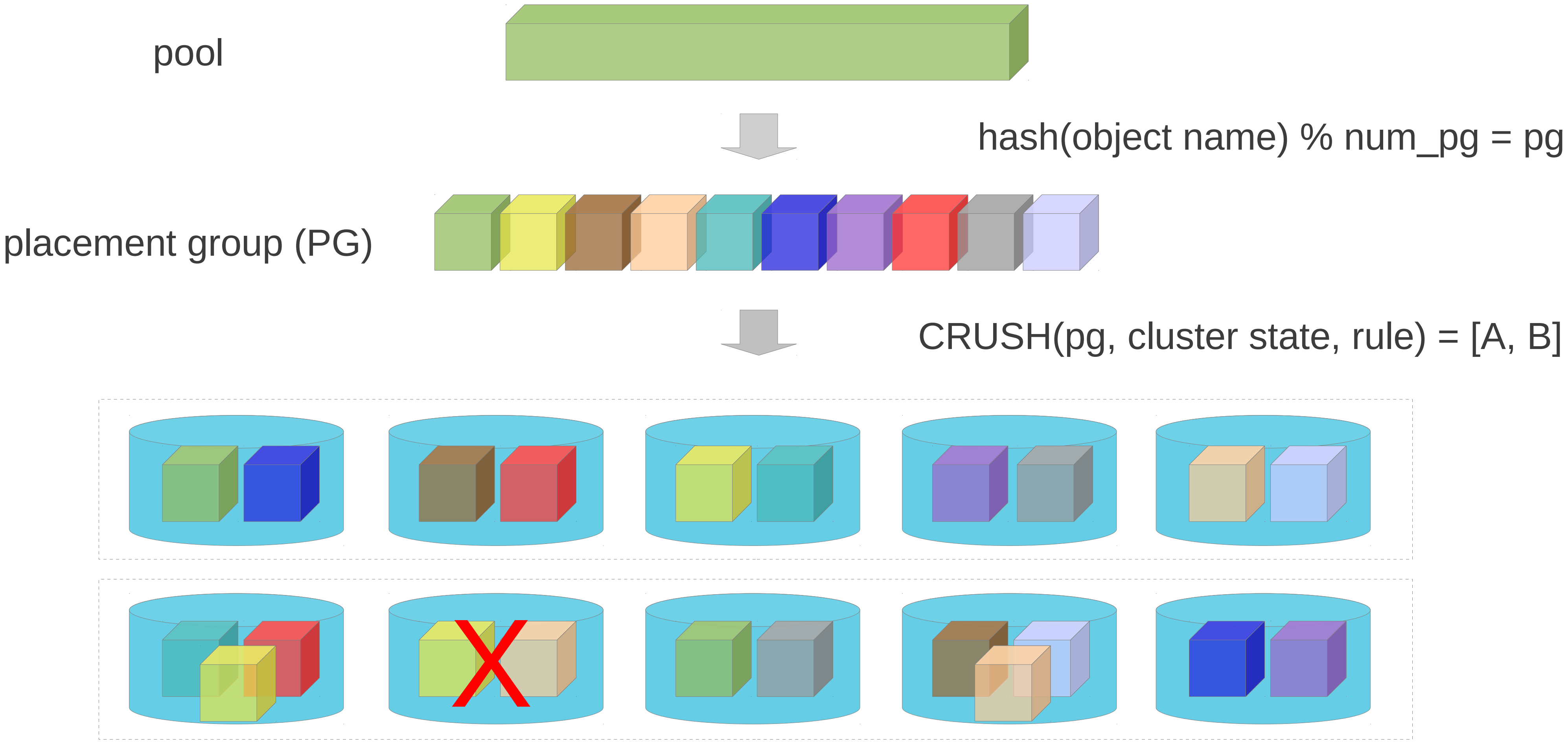
- failure under 2x replication
  - faster rebuild: 1/100th of disk moves from 100 peers to 100 other peers
  - more disks whole subsequent failure would lose data
- independent failures
  - same MTTDL
  - lower E[data loss]
- correlated failures
  - data loss less likely if replicas separated across known failure domains

node types: computation vs stability

- four tree node/bucket types
- varying tradeoffs between
  - computation: speed of choose calculation
  - stability: movement of inputs when bucket items/weights change

Action	**Uniform**	**List**	**Tree**	**Straw**
Speed	O(1)	O(n)	O(log n)	O(n)
Stability (Additions)	Poor	Optimal	Good	Optimal
Stability (Removals)	Poor	Poor	Good	Optimal

object placement



pool

hash(object name) % num_pg = pg

placement group (PG)

CRUSH(pg, cluster state, rule) = [A, B]

declustering

- many to many recovery
  - parallel recovery → fast recovery
  - 1/*n*th as long
  - no bottleneck for individual disks
- no “spare” necessary
  - surviving nodes take up the slack
  - flexible
- cluster is elastic
  - just deploy more storage before it fills up

placement groups

- more means
  - better balancing
  - more metadata, osd peer relationships
  - fully connected cluster
- less means
  - poor balancing
- aim for ~100 per OSD
  - decent utilization variance
  - bounded peers per OSD (~100)

- mkfs time
  - num_osd << pg_bits
- pool creation
  - ceph osd pool create foo 1024
- later
  - eventually adjustable on the fly
  - not upstream yet

- pools are granularity of policy
  - replication count
  - CRUSH placement rule
  - authorization

rados

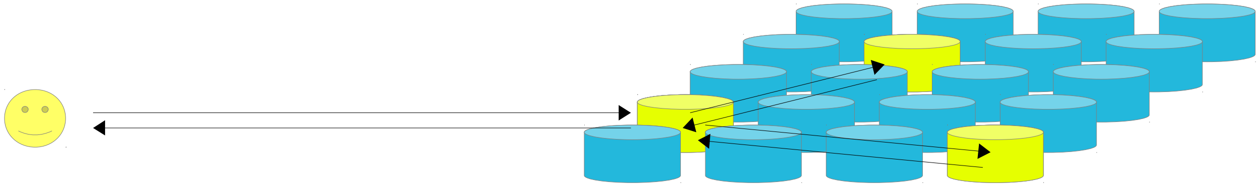
- CRUSH tells us where data should be
- RADOS is responsible for
    - moving it there
    - make sure you can read/write from/to it
    - maintaining illusion of single copy with “consistent” behavior
        - writes are persistent and durable

peering and recovery

- dynamic cluster
  - nodes are added, removed
  - nodes reboot, fail, recover
- “recovery” is the norm
  - “map” records cluster state at point in time
    - ceph-osd node status (up/down, weight, IP)
    - CRUSH function specifying desired data distribution
  - ceph-osds cooperatively migrate data to achieve that
- any map update potentially triggers data migration
  - ceph-osds monitor peers for failure
  - new nodes register with monitor
  - administrator adjusts weights, mark out old hardware, etc.

replication

- all data replicated N times
- ceph-osd cluster handles replication
  - client writes to first replica



  - reduce client bandwidth
  - “only once” semantics
  - cluster maintains strict consistently

rados object API

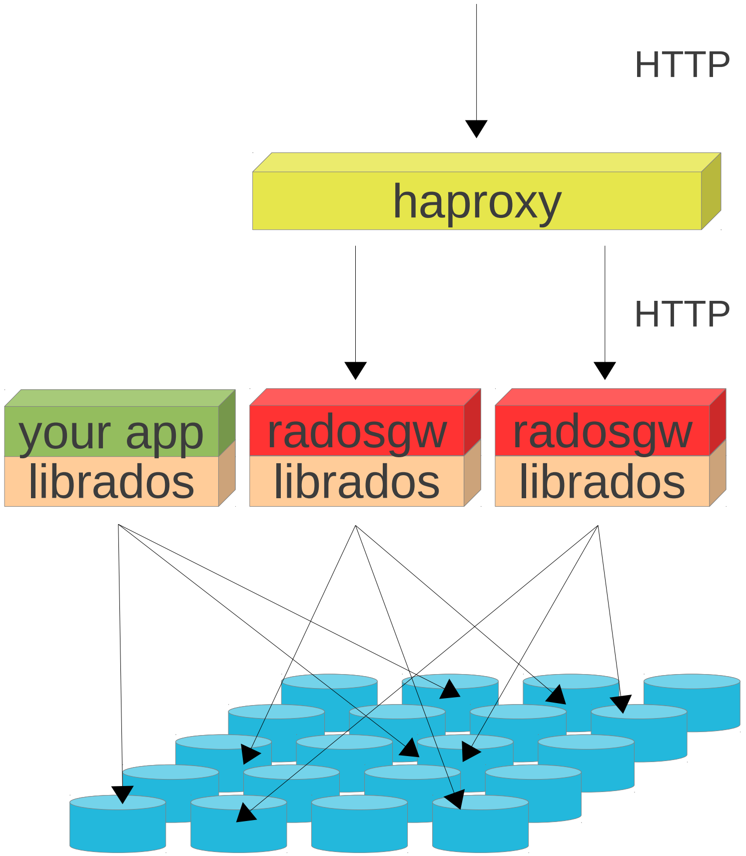
- librados.so
  - C, C++, Python, Java. shell.
- read/write object or byte range, truncate, remove, append
  - like a file
- get/set/remove attr (bytes to KB)
  - based on extended attributes
- get/set/remove key/value (bytes to MB, many keys)
  - based on leveldb
- atomic compound operations/transactions
  - read + getxattr, write + setxattr
  - compare xattr value, if match write + setxattr

rados object API (2)

- per-object snapshot
  - keep multiple read-only past versions of an object
- efficient copy-on-write clone
  - between objects placed in same location in cluster
- classes
  - load new code into cluster to implement new methods
  - calc sha1, grep/filter, generate thumbnail
  - encrypt, increment, rotate image
- watch/notify
  - use object as communication channel between clients

librados, radosgw

- librados
  - direct parallel access to cluster
  - rich API
- radosgw
  - RESTful object storage
    - S3, Swift APIs
  - proxy HTTP to rados
  - ACL-based security for the big bad internet



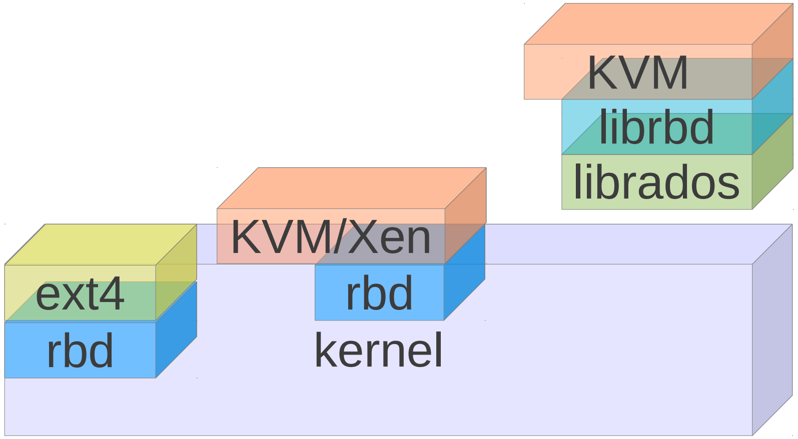
radosgw

- atomic creation/replacement of large objects
- bucket index
  - alphanumerically sorted object listing
  - search by prefix
- ACL security model
  - per-object or per-bucket
- stripe large REST objects over smaller RADOS objects
- use a key/value RADOS object for bucket index
  - efficient query, ordered, etc.
- standalone daemon
  - apache, nginx, lighty
  - fastcgi socket

rados block device (rbd)

rbd – rados block device

- replicated, reliable, high-performance virtual disk
  - striped over objects across entire cluster
  - thinly provisioned, snapshots
  - image cloning (real soon now)
- well integrated
  - Linux kernel driver (/dev/rbd0)
  - qemu/KVM + librbd
  - libvirt, OpenStack
- sever link between virtual machine and host
  - fail-over, live migration



rbd objects

- for each pool used with rbd
  - rbd_info – latest rbd image id
  - rbd_directory – list of images
  - <image>.rbd – image header
    - id
    - size of image, objects
    - snapshots
  - rbd.<id>.<n> – image segments/objects
    - images are sparse

image striping

- disk image striped over power-of-2 byte objects
  - default 4MB objects
  - seek times not significant
  - small enough to be a reasonable IO size
  - small enough to not get too hot
- objects randomly distributed
  - no single (set of) servers responsible for large image
  - workload is well distributed
  - single image can potentially leverage all spindles

rbd and snapshots

- rados clients participate in snapshots
  - provide “context” on write
    - list of snapshots for given object
    - informs copy-on-write behavior on ceph osds
  - clients “watch” header object for changes
- command line tool
  - update header: resize, snap create/delete, rollback
  - notify watchers
- rados class to manage header
  - encapsulate knowledge of on-disk format
  - safe, efficient updates
- snapshot example
  - freeze fs inside VM
    - e.g., xfs_freeze
  - rbd snap create ...
    - update header
    - notify clients
      - re-read headers
  - unfreeze fs

rbd (cont)

- snapshot rollback
  - offline operation
  - repeatable

...

- layering
  - copy-on-write layer over read-only image
  - reads “fall-thru” missing objects
  - writes trigger "copy-up”
- image cloning
  - e.g., OS image for VMs
- image migration
  - create overlay at new location
  - async copy-up
  - sever parent relationship

distributed file system

the metadata problem

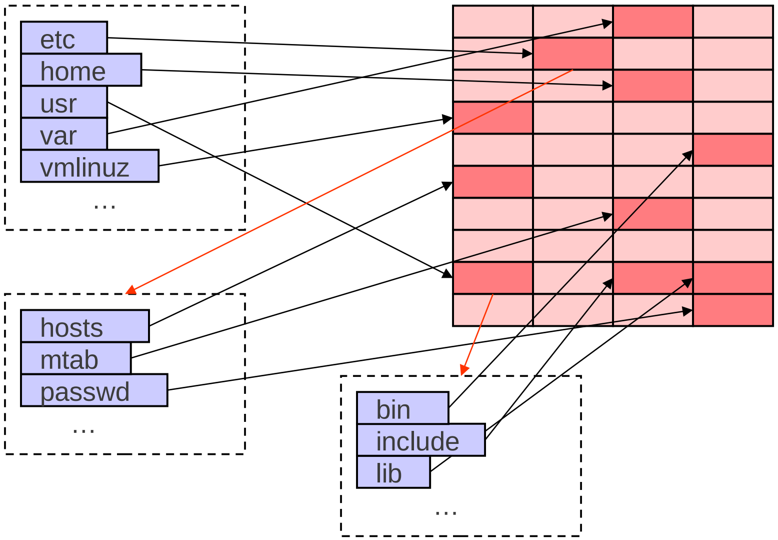
- shared cluster-coherent file system
  - consistent behavior
  - client caching, prefetching
- separate metadata and data paths
  - avoid “server” bottleneck inherent in NFS etc
- dynamic ceph-mds cluster
  - manage file system hierarchy, concurrency
  - redistribute load based on workload
  - leverage object storage infrastructure

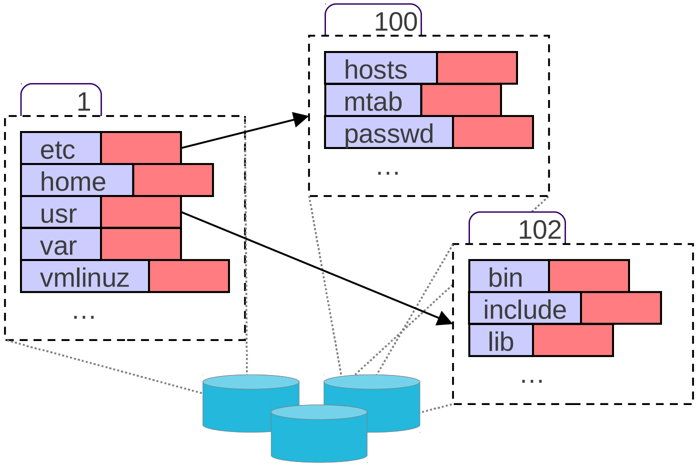
the metadata workload

- most files are small
- most data lives in big files
- most file updates are bursty
  - many metadata updates, then idle
  - untar, compilation
- locality matters
  - intra-directory
  - nearby inter-directory
    - rename
- ls -al
  - readdir + many stats/getattrs
- metadata is critical to performance
  - many small operations, often synchronous

metadata storage

- legacy design is a disaster
  - name → inode → block list → data
  - no inode table locality
  - fragmentation
    - inode table
    - directory

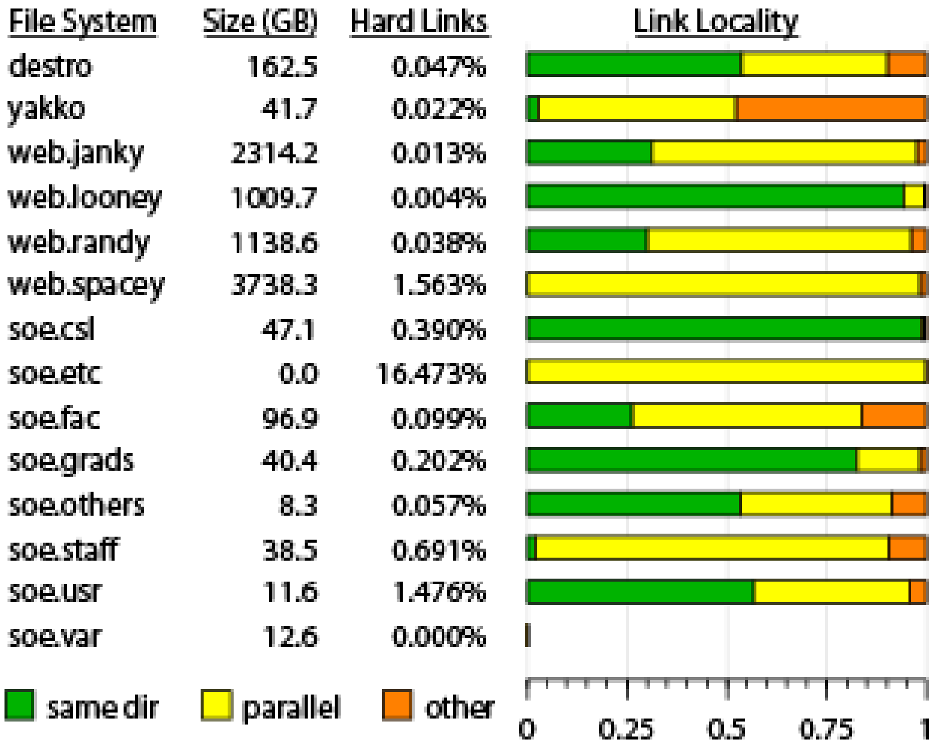




- block lists unnecessary
- inode table mostly useless
  - APIs are path-based, not inode-based
  - no random table access, sloppy caching
- embed inodes inside directories
  - good locality, prefetching
  - leverage key/value objects

hard links?

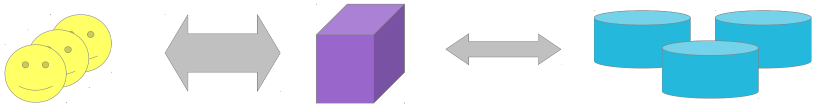
- rare
- useful locality properties
    - intra-directory
    - parallel inter-directory
- “anchor” table provides by-ino lookup
    - degenerates to similar update complexity
    - optimistic read complexity

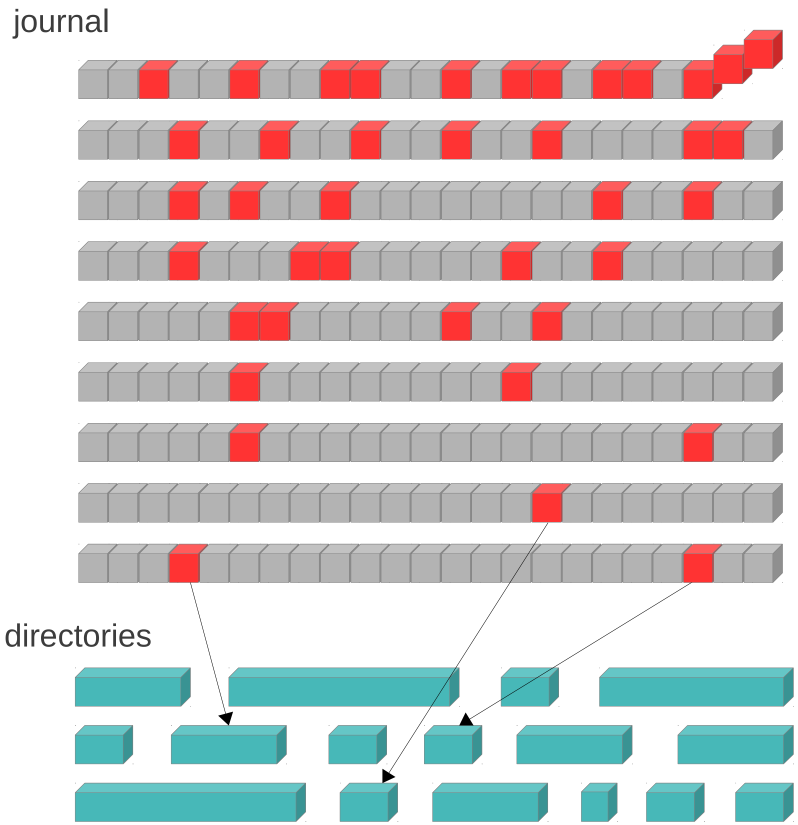


File System	Size (GB)	Hard Links
destro	162.5	0.047%
yakko	41.7	0.022%
web.janky	2314.2	0.013%
web.looney	1009.7	0.004%
web.randy	1138.6	0.038%
web.spacey	3738.3	1.563%
soe.csl	47.1	0.390%
soe.etc	0.0	16.473%
soe.fac	96.9	0.099%
soe.grads	40.4	0.202%
soe.others	8.3	0.057%
soe.staff	38.5	0.691%
soe.usr	11.6	1.476%
soe.var	12.6	0.000%

controlling metadata io

- view ceph-mds as (smart) caching layer
  - reduce reads
    - dir+inode prefetching
  - reduce writes
    - consolidate multiple writes
- large journal or log
  - stripe over objects for efficient io
  - per-segment dirty list, flush to trim
    - combine dir updates over long period
  - two tiers
    - journal for short term
    - per-directory for long term
  - fast failure recovery

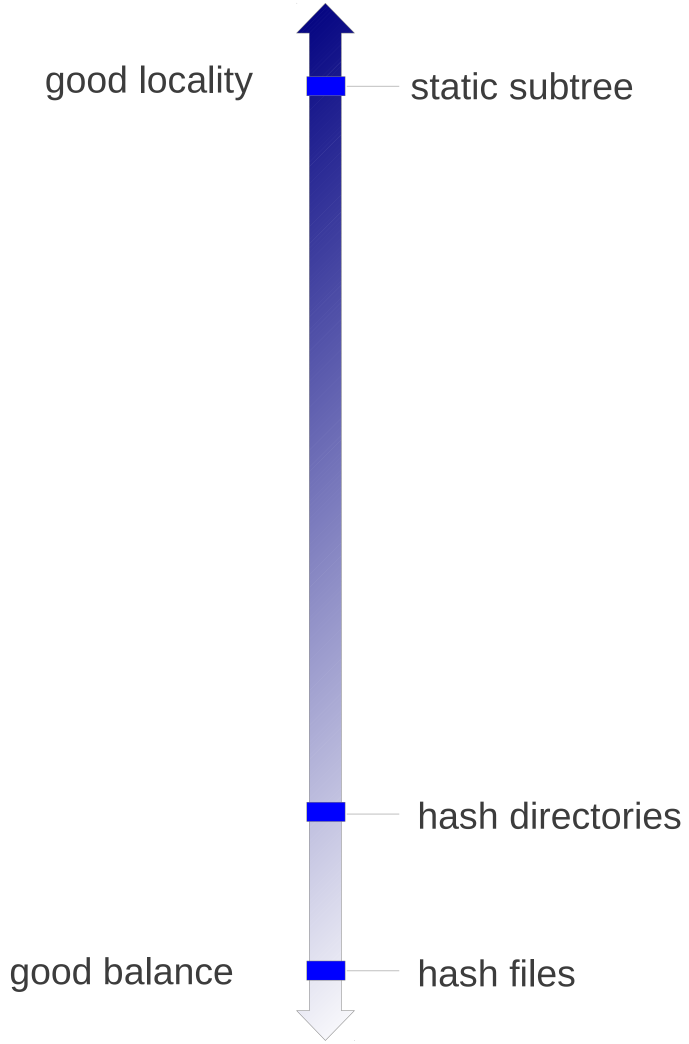




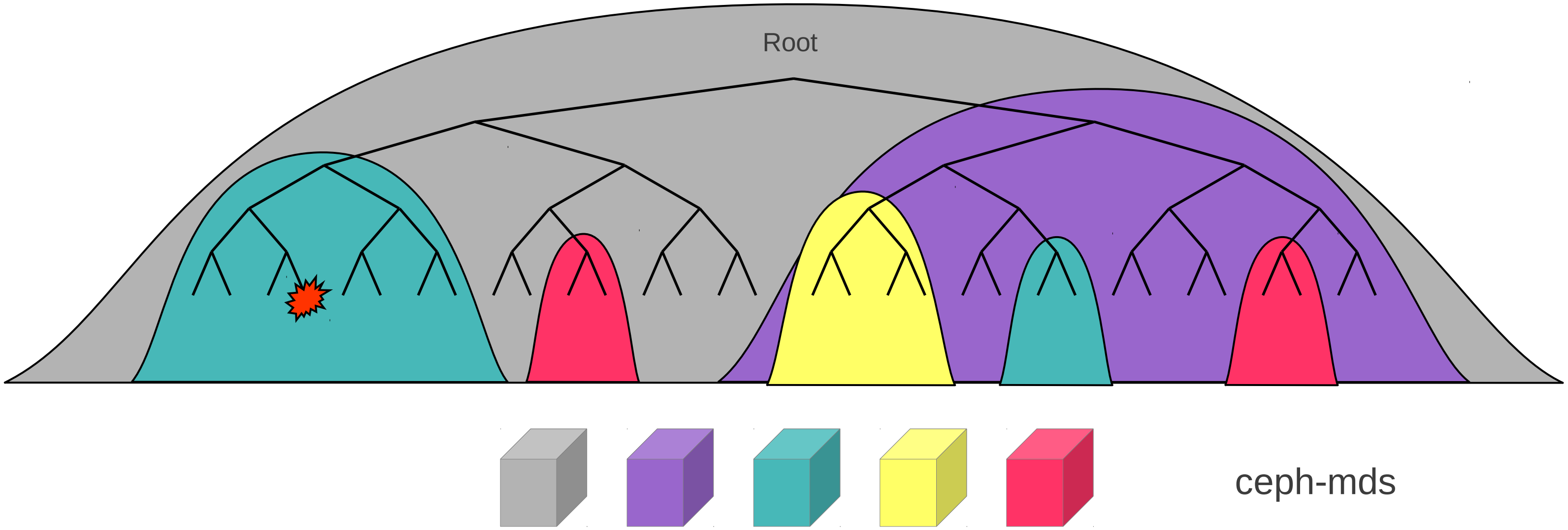
load distribution

- coarse (static subtree)
  - preserve locality
  - high management overhead
- fine (hash)
  - always balanced
  - less vulnerable to hot spots
  - destroy hierarchy, locality

- can a dynamic approach capture benefits of both extremes?

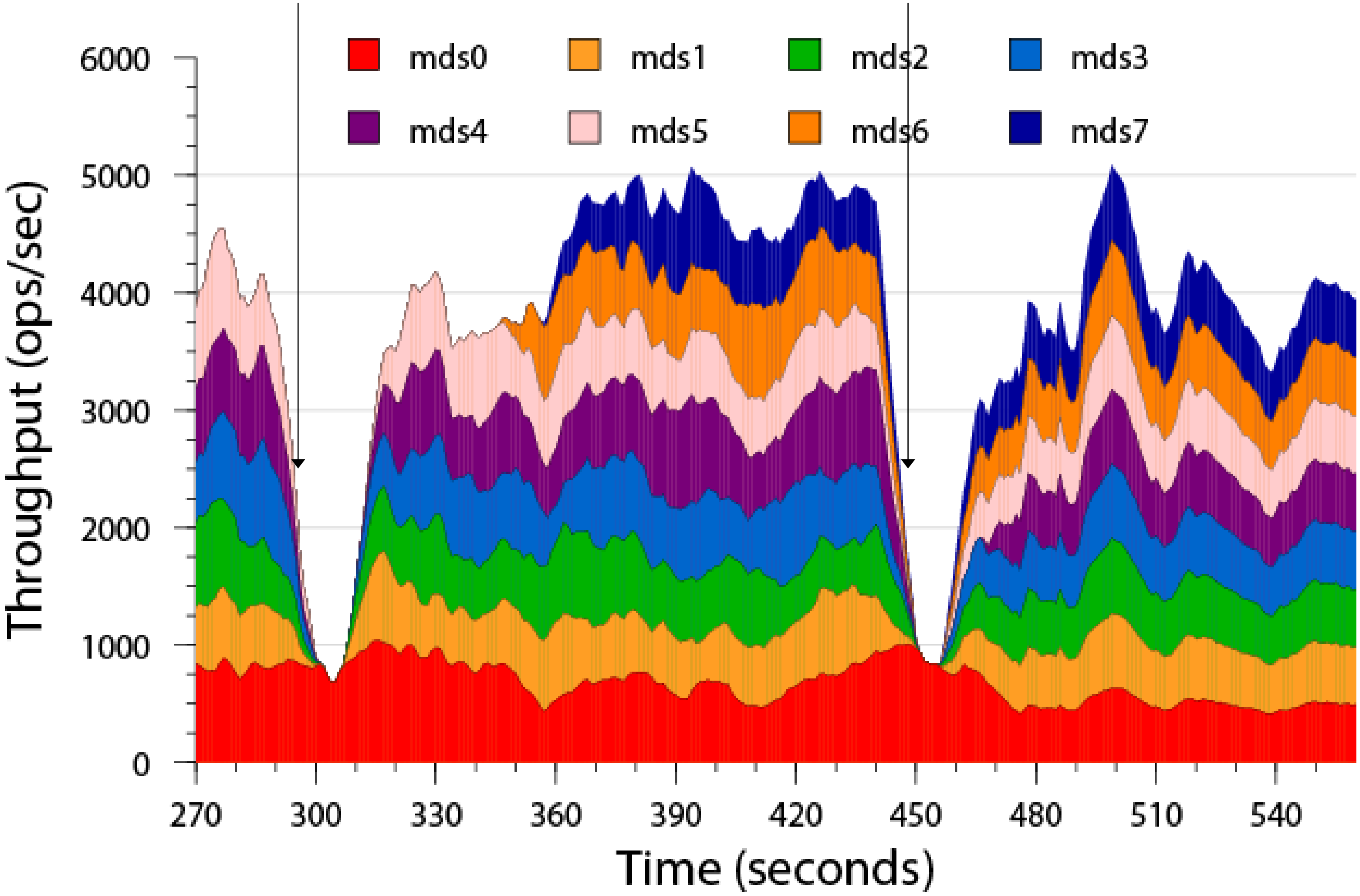


dynamic subtree partitioning



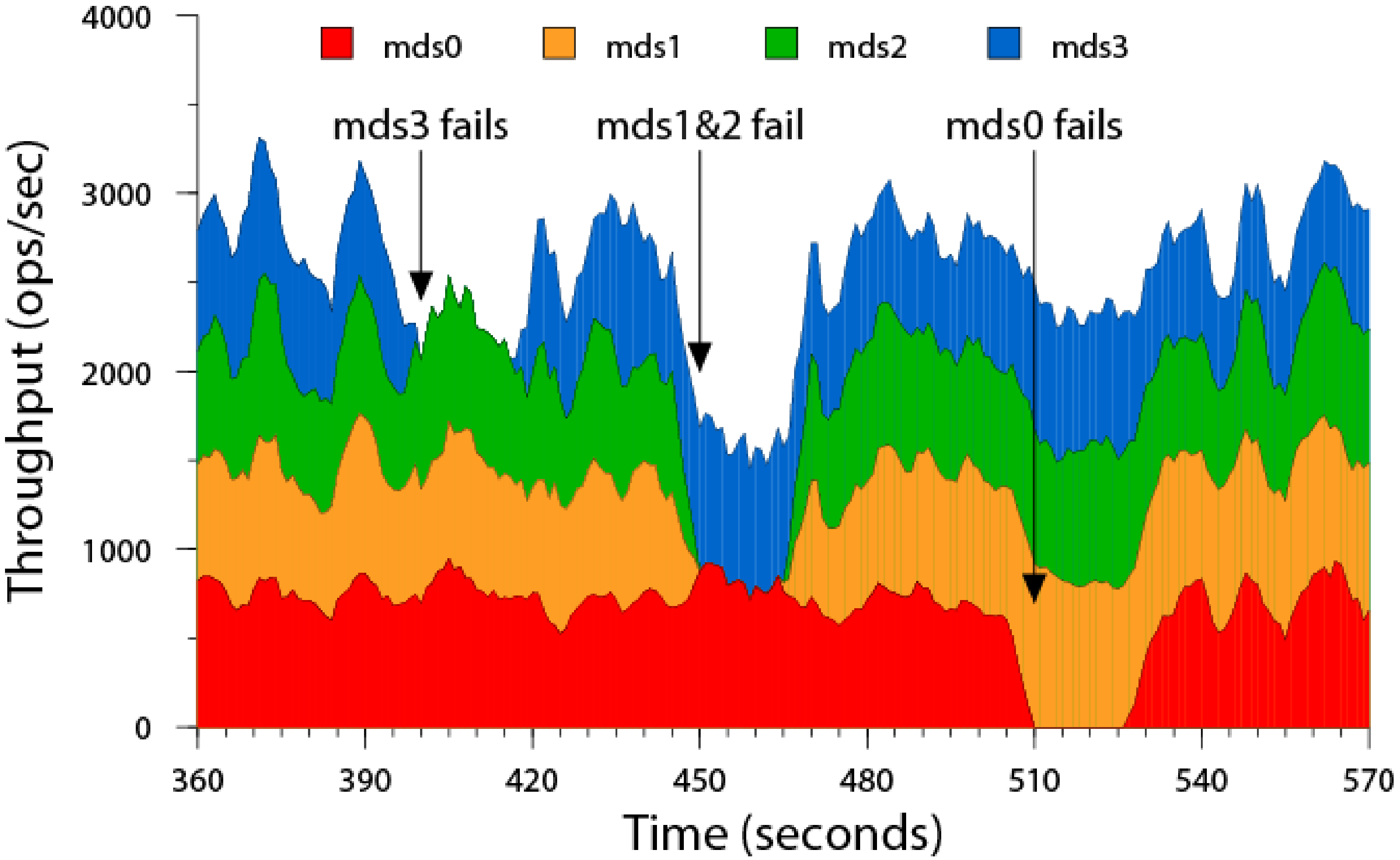
- efficient
  - hierarchical partition preserve locality
  - single mds for any piece of metadata
- adaptive
  - move work from busy to idle servers
  - hot metadata gets replicated
- scalable
  - arbitrarily partition metadata
  - coarse when possible, fine when necessary
- dynamic
  - daemons can join/leave
  - take over for failed nodes

workload adaptation


many directories

same directory

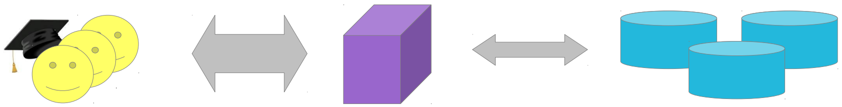
failure recovery



client protocol

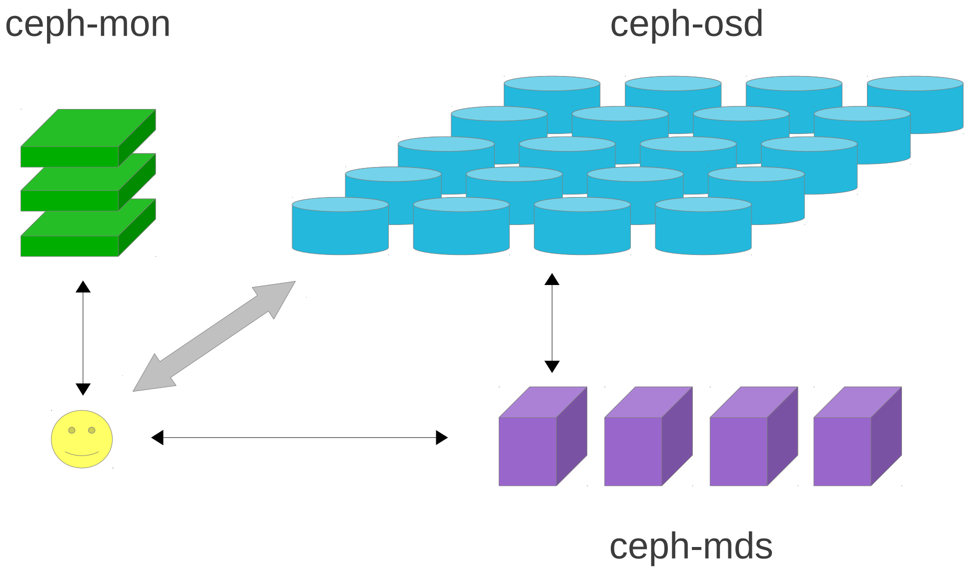
- stateless protocols
  - either inefficient...
    - all operations synchronous
  - ...or inconsistent
    - e.g. NFS, timeout based caching
- stateful protocols
  - complex
  - even more complex recovery
  - do well in non-failure

- we choose stateful
  - consistent caches
  - aggressive prefetching
- async whenever possible
  - consistency vs durability
- fine-grained metadata locks/leases
  - size/mtime vs mode/uid/gid



an example

- `mount -t ceph 1.2.3.4:/ /mnt`
  - 3 ceph-mon RT
  - 2 ceph-mds RT (1 ceph-mds to -osd RT)
- `cd /mnt/foo/bar`
  - 2 ceph-mds RT (2 ceph-mds to -osd RT)
- `ls -al`
  - open
  - readdir
    - 1 ceph-mds RT (1 ceph-mds to -osd RT)
  - stat each file
  - close
- `cp * /tmp`
  - N ceph-osd RT



recursive accounting

- ceph-mds tracks recursive directory stats
  - file sizes
  - file and directory counts
  - modification time
- virtual xattrs present full stats
- clean, efficient implementation
  - metadata lives in a hierarchy
  - lazy propagation of changes up the tree

```
$ ls -alSh | head
total 0
drwxr-xr-x 1 root            root      9.7T 2011-02-04 15:51 .
drwxr-xr-x 1 root            root      9.7T 2010-12-16 15:06 ..
drwxr-xr-x 1 pomceph         pg4194980 9.6T 2011-02-24 08:25 pomceph
drwxr-xr-x 1 mcg_test1       pg2419992  23G 2011-02-02 08:57 mcg_test1
drwx--x--- 1 luko            adm        19G 2011-01-21 12:17 luko
drwx--x--- 1 eest            adm        14G 2011-02-04 16:29 eest
drwxr-xr-x 1 mcg_test2       pg2419992 3.0G 2011-02-02 09:34 mcg_test2
drwx--x--- 1 fuzyceph        adm       1.5G 2011-01-18 10:46 fuzyceph
drwxr-xr-x 1 dallasceph      pg275     596M 2011-01-14 10:06 dallasceph
```

snapshots

- volume or subvolume snapshots unusable at petabyte scale
  - snapshot arbitrary subdirectories
- simple interface
  - hidden '.snap' directory
  - no special tools

```
$ mkdir foo/.snap/one    # create snapshot
$ ls foo/.snap
one
$ ls foo/bar/.snap
_one_1099511627776       # parent's snap name is mangled
$ rm foo/myfile
$ ls -F foo
bar/
$ ls -F foo/.snap/one
myfile  bar/
$ rmdir foo/.snap/one    # remove snapshot
```

practical private cloud setup

hardware deployment

- commodity
  - SAS/SATA, HDD/SDD
  - ethernet (IP)
  - NVRAM
- user-level daemons
  - mon
    - lightweight, some local disk space
  - osd
    - big backend filesystem, preferably btrfs
    - fast journal (SSD, NVRAM)
  - mds
    - no disk
    - lots of RAM
- RAID
  - more reliable
  - local recovery
  - some storage overhead
- JBOD
  - no overhead
  - network recovery
  - some software fault isolation
- tend to prefer JBOD, currently
  - osd per disk
  - shared SSD for journals

installation

- git, tarball, deb, rpm
- debs are easiest
  - debian sid, wheezy, squeeze
  - ubuntu precise, oneiric, maverick

- rpms
  - open build service
  - Fedora, RHEL/CentOS
  - OpenSUSE, SLES

- add apt source

```
echo deb http://ceph.newdream.net/debian precise main > /etc/apt/sources.list.d/ceph.list
```

- install

```
apt-get install ceph
apt-get install librbd1, librados2, libcephfs1
apt-get install radosgw
```

cluster configuration

- /etc/ceph/ceph.conf
  - ini-style config file
  - section per daemon
  - inherit type/global sections
  - daemon behavior; no cluster info
- past/present
  - can be global
  - enumerates daemons
  - daemon start/stop when host field matches hostname
- future
  - udev hooks
  - chef, juju, etc.

```
[global]
    auth supported = cephx

[mon]
    mon data = /var/lib/ceph/ceph-mon.$id

[mon.a]
    host = mymon-a
    mon addr = 1.2.3.4:6789
[mon.b]
    host = mymon-b
    mon addr = 1.2.3.5:6789
[mon.c]
    host = mymon-c
    mon addr = 1.2.3.4:6789

[osd]
    osd data = /var/lib/ceph/ceph-osd.$id

[osd.0]
    host = myosd0
```

creating a cluster

- easiest
  - set up ssh keys
  - *mkcephfs -c conf -a –mkbtrfs*
  - distribute admin key
- start up
  - *service ceph start*

- *ceph* command
  - monitoring, status
  - admin
- *ceph health*
  - HEALTH_OK
  - HEALTH_WARN …
- *ceph -w*
  - watch cluster state change

cluster management

- ceph command-line tool
  - uses client.admin user to communicate with monitors
- admin-friendly text and script-friendly json

```
ceph osd dump
ceph osd dump –format=json
ceph health
```

authentication and authorization

- design based on kerberos
- monitors are trusted authority
  - maintain repository of secret keys
  - clients and daemons
    - authenticate against ceph-mon
    - mutual authentication (authenticity of server confirmed)
    - get a ticket with a signed/encrypted capability
      - set of (service type, opaque blob) pairs
- daemons authenticate on TCP connection open
  - limit access based on signed capability
  - e.g., a librados client “client.foo” may have capability
    - osd = “allow rwx pool=foo, allow r pool=bar”
  - current capability definitions coarse; can be refined

- ceph command defaults to client.admin, key in /etc/ceph/keyring
  - -n <name> to set “user”
  - -k <keyring path>
- keys and associated capabilities registered with the monitor
  - ceph auth add …
  - ceph auth list

...

ceph-osd failure

- kill a ceph-osd daemon
  - peers will discover failure
  - monitor will update osdmap
  - cluster will repeer
  - degraded cluster
- mark failed nodes out
  - make CRUSH skip them
  - data remapped to new nodes
  - cluster will “recover” (re-replicate/migrate data)
- configurable timeouts

```
killall ceph-osd
service stop osd.12
```

```
ceph osd out 12
```

ceph-osd recovery

- restart daemon
  - comes back up...
  - not auto-marked in unless it was auto-marked out
  - optional behavior for new nodes
    - admin or deployment driven migration

add new osd

- ceph osd create

```
12
```

- add to ceph.conf

```
[osd.12]
    host = plana12
    btrfs devs = /dev/sdb
```

- mkfs + mount

```
mkfs.btrfs /dev/sdb
mkdir -p /var/lib/ceph/osd-data/12
mount /dev/sdb /var/lib/ceph/osd-data/12
ceph-osd –mkfs -i 12 –mkkey
```

- add auth key

```
ceph auth add osd.12 osd 'allow *' mon 'allow
rwx' -i /var/lib/ceph/osd-data/12/keyring
```

- start

```
service ceph start osd.12
```

- osd part of cluster, but stores no data
- add to crush map

```
ceph osd tree
ceph osd crush add 12 osd.12 1.0
host=plana12 rack=unknownrack
pool=default
ceph osd tree
```

- data migration starts

a

adjusting device weights

- ceph osd tree
  - show crush hierarchy, weights
- ceph osd crush reweight osd.12 .7
  - adjust crush weight
  - will trigger data migration

modifying crush map

- extract map

  ```
  ceph osd getcrushmap -o cm
  crushtool -d cm -o cm.txt
  ```

- modify
- inject new map

  ```
  crushtool -c cm.txt -o cm.new
  ceph osd setcrushmap -i cm.new
  ```

crush map

```
# begin crush map

# devices
device 0 osd.0

# types
type 0 osd
type 1 host
type 2 rack
type 3 pool
```

```
# buckets
host localhost {
        id -2
        # weight 1.000
        alg straw
        hash 0  # rjenkins1
        item osd.0 weight 1.000
}
rack localrack {
        id -3
        # weight 1.000
        alg straw
        hash 0  # rjenkins1
        item localhost weight 1.000
}
pool default {
        id -1
        # weight 1.000
        alg straw
        hash 0  # rjenkins1
        item localrack weight 1.000
}
```

crush rules

```
# rules
rule data {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take default
        step choose firstn 0 type osd
        step emit
}
rule metadata {
        ruleset 1
        type replicated
        min_size 1
        max_size 10
        step take default
        step choose firstn 0 type osd
        step emit
}
```

adjust replication

- pool “size” is replication level

```
ceph osd dump | grep ^pool
```

- just another osdmap change

```
ceph osd pool rbd set data size 3
```

rbd example

- create an rbd user

```
ceph-authtool --create-keyring -n client.rbd –gen-key rbd.keyring
```

```
ceph auth add client.rbd osd “allow *” mon “allow *” -i rbd.keyring
```

- import an image

```
rbd import precise-server.img foo
```

- take an initial snapshot

```
rbd snap create –snap=orig foo
```

install libvirt, qemu

- apt source

  ```
  echo deb http://ceph.newdream.net/debian precise main > /etc/apt/sources.list.d/ceph.list
  ```

- apt-get install libvirt kvm

libvirt authentication

- include rbd secret in libvirt keyring

```
<secret ephemeral="no" private="no">
 <uuid>fe1447b4-9959-d104-b902-8cf6bf540a5c</uuid>
 <usage type="ceph">
  <name>client.rbd secret</name>
 </usage>
</secret>
```

```
virsh secret-define secret.xml
```

```
virsh secret-set-value <uuid> `ceph-authtool -p
rbd.keyring -n client.rbd`
```

define virtual machine

- reference rbd backend disk

```
<disk type="network" device="disk">
 <driver name="qemu" type="raw"/>
 <auth username="rbd">
  <secret type="ceph" usage="client.rbd secret"/>
 </auth>
 <source protocol="rbd" name="rbd/foo">
  <host name="10.214.131.38" port="6789"/>
  <host name="10.214.131.37" port="6789"/>
  <host name="10.214.131.35" port="6789"/>
 </source>
 <target dev="vda" bus="virtio"/>
 <address type="pci" domain="0x0000" bus="0x00" slot="0x04" function="0x0"/>
</disk>
```

```
virsh define ubuntu-on-rbd.xml
```

resize, rollback image

- we can expand/contract images

```
rbd resize –size 20000 foo
rbd info foo
rbd resize –size 10000 foo
```

- if the image goes bad (e.g., rm -rf /)

```
rbd snap rollback –snap=orig foo
```

live migration

- define identical image on two libvirt hosts
  - same xml with same backend disk
- trigger KVM migration via libvirt

```
virsh migrate --live foo qemu+ssh://target/system
```

- very easy with virt-manager gui

why

- limited options for scalable open source storage
  - lustre
  - gluster
  - HDFS
  - Orange
- proprietary solutions
  - marry hardware and software
  - few scale out
- industry needs open alternatives

project status

- 12 developers
- 4 business, community, support
- rados, rbd, rgw supported
- distributed file system next
- included in
  - mainline kernel
  - linux distros (debian, ubuntu, fedora, suse)

why we like btrfs

- pervasive checksumming
- snapshots, copy-on-write
- efficient metadata (xattrs)
- inline data for small files
- transparent compression
- integrated volume management
  - software RAID, mirroring, error recovery
  - SSD-aware
- online fsck
- active development community