



Ceph distributed storage

Sage Weil
new dream network /
DreamHost

What is Ceph?

- ❑ Scalable storage system
 - ❑ 1 to 1000s of nodes
 - ❑ Gigabytes to exabytes
- ❑ Reliable
 - ❑ No single points of failure
 - ❑ All data is replicated
 - ❑ Self-healing
- ❑ Self-managing
 - ❑ Automatically (re)distributes stored data

Ceph storage services

- ❑ Ceph distributed file system
 - ❑ POSIX, snapshots
- ❑ RBD: rados block device
 - ❑ Thinly provisioned, snapshottable virtual disk
 - ❑ Linux kernel driver; Native support in Qemu/KVM
- ❑ radosgw: RESTful object storage proxy
 - ❑ S3 and Swift compatible interfaces
- ❑ librados: native object storage
 - ❑ Fast, direct access to storage cluster
 - ❑ Flexible: pluggable object classes

What makes it different?

- ❑ Open source
 - ❑ LGPL2, open development, shared copyright
- ❑ Flexible
 - ❑ Commodity hardware, incremental scalability
 - ❑ Modular server components
- ❑ Multiple storage APIs
- ❑ Avoid legacy design antipatterns
 - ❑ Block-based interfaces, manual workload partition, passive servers

Outline

- Objects
- Where to store data objects
 - And how to find them again
- Making the object storage self-managing
 - Make good use of processors on storage nodes
 - Extensible object abstraction
- Building a distributed POSIX file system
 - Adaptive and scalable

Object storage

❑ Objects

- ❑ Alphanumeric name
- ❑ Data blob (bytes to gigabytes)
- ❑ Named attributes (foo=bar)

❑ Object pools

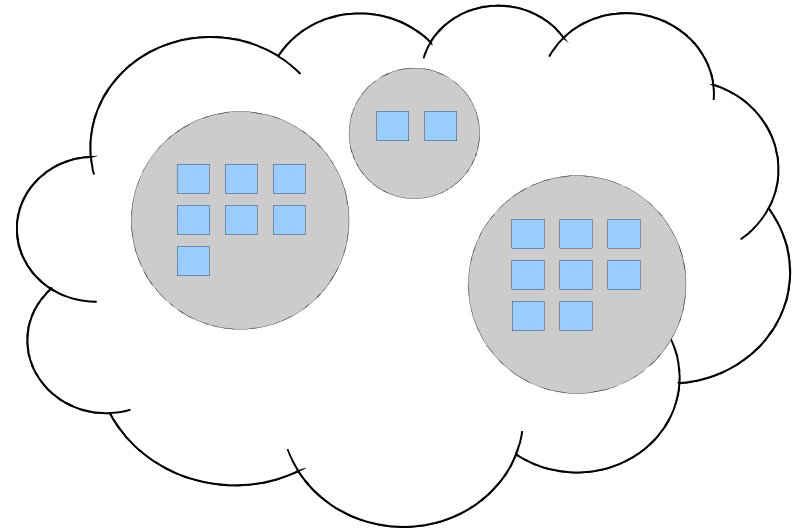
- ❑ Separate flat namespace

❑ Cluster of servers store all objects

- ❑ RADOS: **R**eliable **a**utonomic **d**istributed **o**bject **s**tore

❑ Low-level storage infrastructure

- ❑ librados, radosgw
- ❑ RBD, Ceph distributed file system



Where is my data?

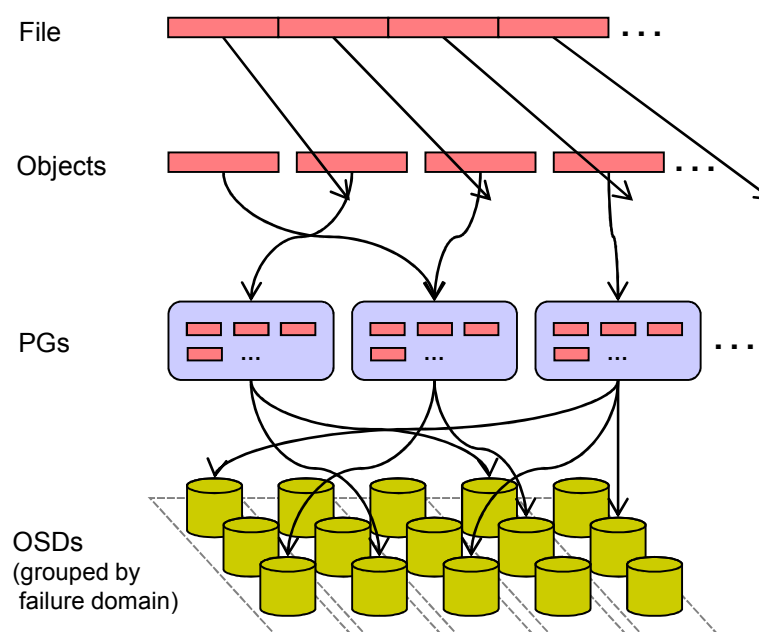
- ❑ Remember where you put it
 - ❑ "I stored A on node B in file C"
 - ❑ System can't move data: you won't be able to find it
- ❑ Name object; system tracks data location
 - ❑ Ask location server where A is, then ask storage node for the data
 - ❑ System can move A, update its directory
- ❑ Calculate pseudo-random mapping
 - ❑ name + current cluster members → current location
 - ❑ No directory necessary

Placement with CRUSH

- ❑ **Functional:** $x \rightarrow [node\ 12, node\ 34]$
 - ❑ Pseudo-random, uniform (weighted) distribution
- ❑ **Fast:** $O(\log n)$ calculation
- ❑ **Stable:** adding devices remaps few x 's
- ❑ **Reliable:** separate replicas
 - ❑ Describe cluster in terms of physical infrastructure
 - ❑ e.g., devices, servers, cabinets, rows, DCs, etc.
 - ❑ "three replicas, different cabinets, same row"

Ceph data placement

- Files/devices striped over objects
 - 4 MB objects by default
- Objects mapped to *placement groups* (PGs)
 - $\text{pgid} = \text{hash}(\text{object}) \ \& \ \text{mask}$
- PGs mapped to sets of OSDs
 - $\text{crush}(\text{cluster}, \text{rule}, \text{pgid}) = [\text{osd2}, \text{osd3}]$
 - ~100 PGs per node
 - Pseudo-random, statistically uniform distribution



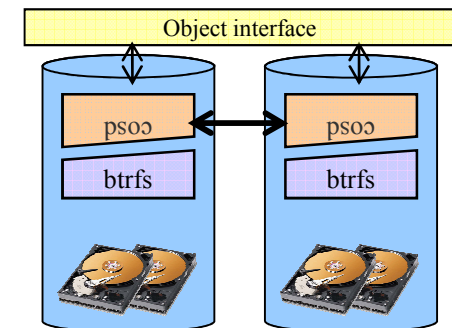
- **Fast**– $O(\log n)$ calculation, no lookups
- **Reliable**– replicas span failure domains
- **Stable**– adding/removing OSDs moves few PGs

What is a storage cluster?

- ❑ Yesteryear
 - ❑ SAN: FC network
 - ❑ LUNs, or SAN file systems
 - ❑ Expensive and antiquated
- ❑ Today
 - ❑ NAS: iSCSI, NFS, CIFS over IP
 - ❑ Storage (SSDs, HDDs) deployed in rackmount shelves with CPU, memory, NIC, RAID...
- ❑ But storage servers are still *passive*...

Ceph storage servers

- ❑ Ceph storage nodes (OSDs)
 - ❑ **cosd** object storage daemon
 - ❑ btrfs volume of one or more disks
- ❑ *Actively* collaborate with peers
 - ❑ Replicate data (n times—admin can choose)
 - ❑ Consistently apply updates
 - ❑ Detect node failures
 - ❑ Migrate data



Functional object placement

- ❑ **OSD map** completely specifies data placement
 - ❑ OSD cluster membership and state (up/down etc.)
 - ❑ CRUSH function mapping objects → PGs → OSDs
- ❑ OSDs can act intelligently – everyone knows and agrees where objects belong
 - ❑ Coordinate writes with replica peers
 - ❑ Copy or migrate objects to proper location

OSD peering and recovery

- ❑ **cosd** will "peer" on startup or map change
 - ❑ Contact other replicas of PGs they store
 - ❑ Ensure PG contents are in sync, and stored on the correct nodes
- ❑ Identical, robust process for *any* map change
 - ❑ Node failure
 - ❑ Cluster expansion/contraction
 - ❑ Change in replication level
- ❑ Self-healing, self-managing

Why btrfs?

- ❑ Featureful
 - ❑ COW, snapshots, checksums, multi-device, compression, online scrub/repair, tasty name
- ❑ Leverage internal transactions, snapshots
 - ❑ OSDs need consistency points for sane recovery
- ❑ Hooks into copy-on-write infrastructure
 - ❑ Clone data content between files (objects)
- ❑ Ext[34], XFS, etc. can also work...
 - ❑ Inefficient snapshots, journaling

Object storage API

- ❑ librados
 - ❑ Direct, parallel access to entire OSD cluster
 - ❑ Bazillions of objects across thousands of nodes
 - ❑ C, C++, Python, Ruby, Java, PHP bindings
- ❑ Compound operations
 - ❑ Read object extent, read attr
 - ❑ Cmpxattr guard, truncate, write extent, set xattr
- ❑ Snapshot primitives
 - ❑ Client provided 'context' informs COW decisions

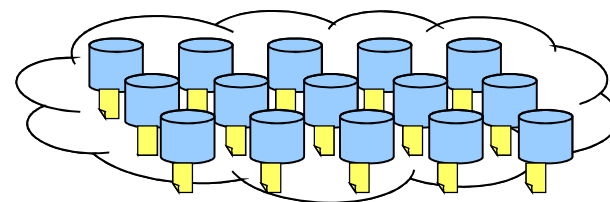
Object storage API (2)

- ❑ Watch/notify infrastructure
 - ❑ Client register interest (watch) on objects
 - ❑ Persistent across a session (adjustable timeouts)
 - ❑ Clients can publish to watchers (notify)
 - ❑ Used for cooperative locking, message passing
- ❑ Key/value interface
 - ❑ Treat object data as key/value bundle
- ❑ Extensible!

Object classes

❑ Start with basic object methods

- ❑ {read, write, zero} extent; truncate
- ❑ {get, set, remove} attribute
- ❑ delete



❑ Dynamically loadable object classes

- ❑ Implement new methods based on existing ones
- ❑ e.g. "calculate SHA1 hash," "rotate image," "invert matrix",
"update this blob of JSON/XML," etc.

❑ Moves computation to data

- ❑ Avoid read/modify/write cycle over the network
- ❑ e.g., MDS uses simple key/value methods to update objects
containing directory content

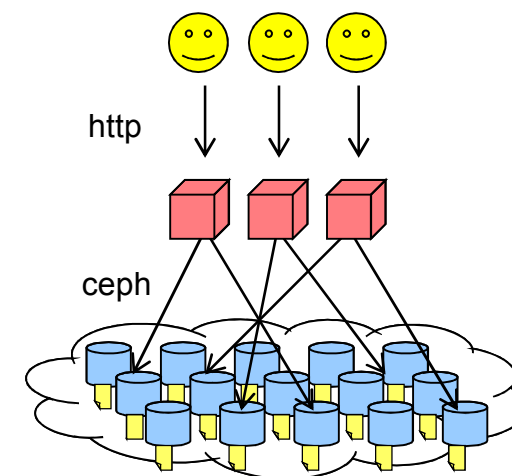
RESTful object storage

□ radosgw

- HTTP RESTful gateway
- S3, Swift protocols

□ Uses

- libfcgid (fastcgi) – webserver agnostic
- librados
- Clones data between objects for atomic GET/PUT and object versioning
- Watch/notify to coordinate cache invalidation across radosgw process



RBD: Rados Block Device

- ❑ Virtual disk image striped over objects
 - ❑ Reliable shared storage (librados)
 - ❑ VM migration between hosts
- ❑ Thinly provisioned
- ❑ Per-image snapshots
 - ❑ Watch/notify for synchronization between clients
- ❑ Layering (WIP)
 - ❑ Copy-on-write overlay over existing 'gold' image
 - ❑ Fast image creation or migration

RBD: Rados Block Device

❑ Native Qemu/KVM support (via librbd)

```
$ qemu-img create -f rbd rbd:mypool/myimage 10G  
$ qemu-system-x86_64 --drive format=rbd,file=rbd:mypool/myimage
```

❑ Linux kernel driver (2.6.37+)

```
$ echo "1.2.3.4 name=admin mypool myimage" > /sys/bus/rbd/add  
$ mke2fs -j /dev/rbd0  
$ mount /dev/rbd0 /mnt
```

❑ libvirt, openstack

❑ Qemu or kernel rbd mapping

❑ CLI (via librbd)

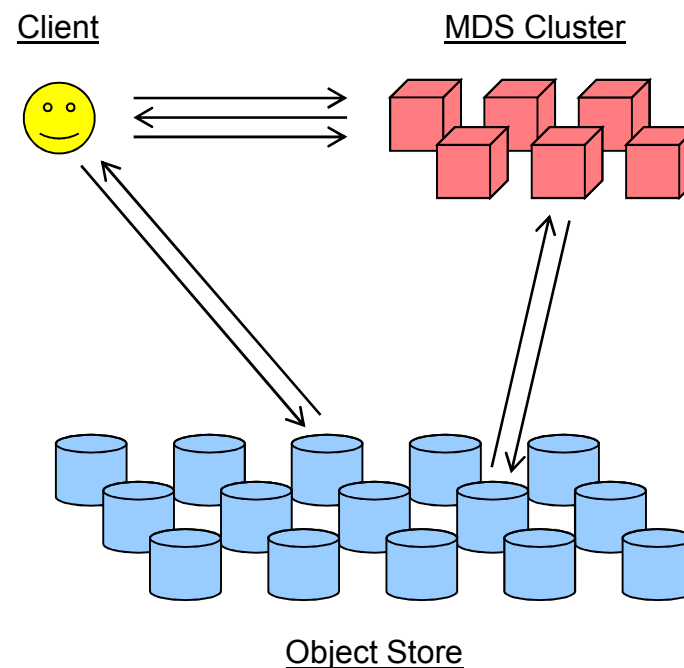
```
$ rbd create foo --size 20G  
$ rbd list  
foo  
$ rbd snap create --snap=asdf foo  
$ rbd resize foo --size=40G  
$ rbd snap create --snap=qwer foo  
$ rbd snap ls foo  
2   asdf   20971520  
3   qwer   41943040
```

Ceph distributed file system

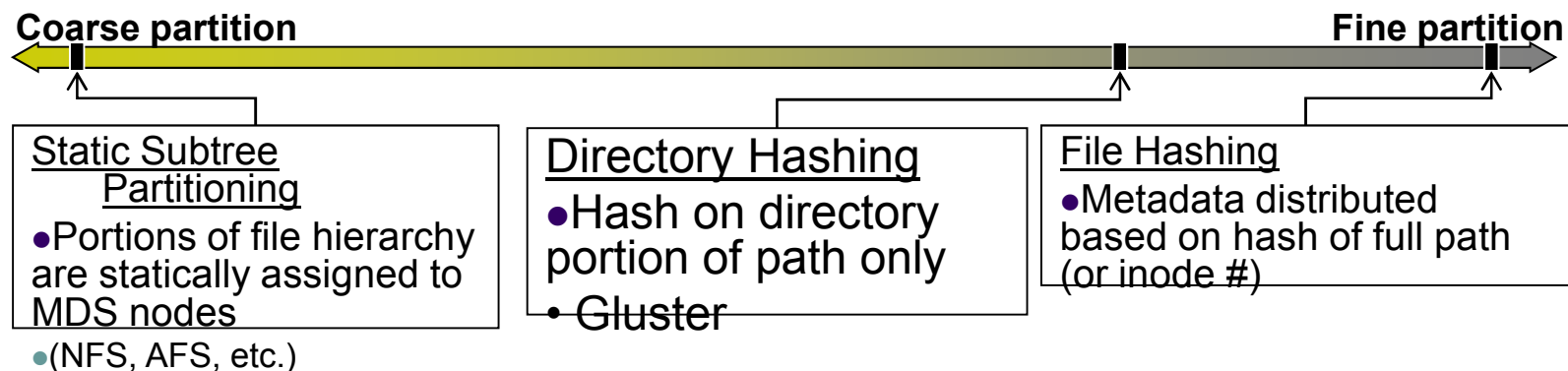
- ❑ Create POSIX file system on top of objects
- ❑ Cluster of **cmds** daemons
 - ❑ No local storage – all metadata stored in objects
 - ❑ Lots of RAM – function has a large, distributed, coherent cache arbitrating file system access
- ❑ Dynamic cluster
 - ❑ New daemons can be started up willy nilly
 - ❑ Automagically load balanced

A simple example

- ❑ `fd=open("/foo/bar", O_RDONLY)`
 - ❑ Client: requests open from MDS
 - ❑ MDS: reads directory /foo from object store
 - ❑ MDS: issues capability for file content
- ❑ `read(fd, buf, 1024)`
 - ❑ Client: reads data from object store
- ❑ `close(fd)`
 - ❑ Client: relinquishes capability to MDS
 - ❑ MDS out of I/O path
 - ❑ Object locations are well known—calculated from object name



Partitioning metadata



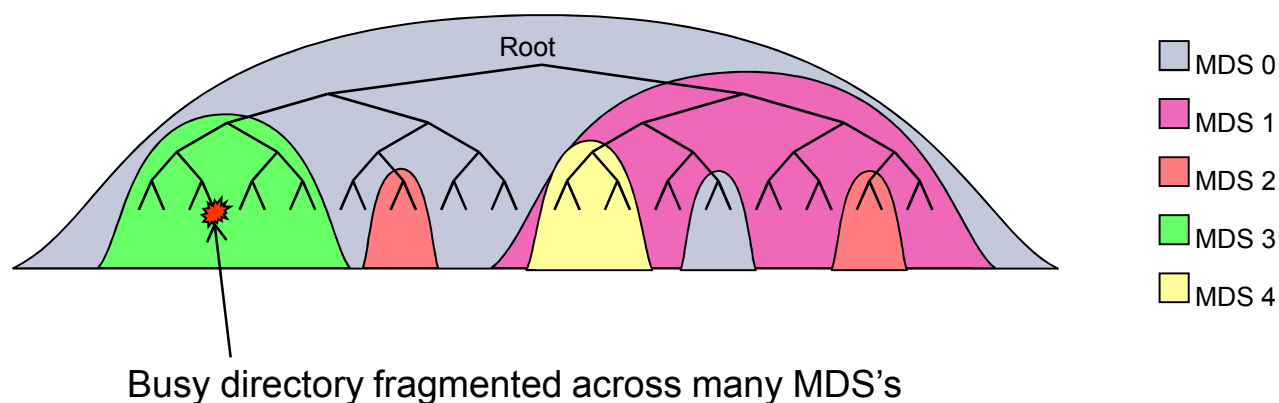
❑ Coarse distribution (static subtree partitioning)

- ❑ Hierarchical partition preserves locality
- ❑ High management overhead

❑ Fine distribution (directory or file hashing)

- ❑ Better load balance, fewer hot spots
- ❑ Destroys locality (ignores hierarchy)

Dynamic subtree partitioning



□ Scalable

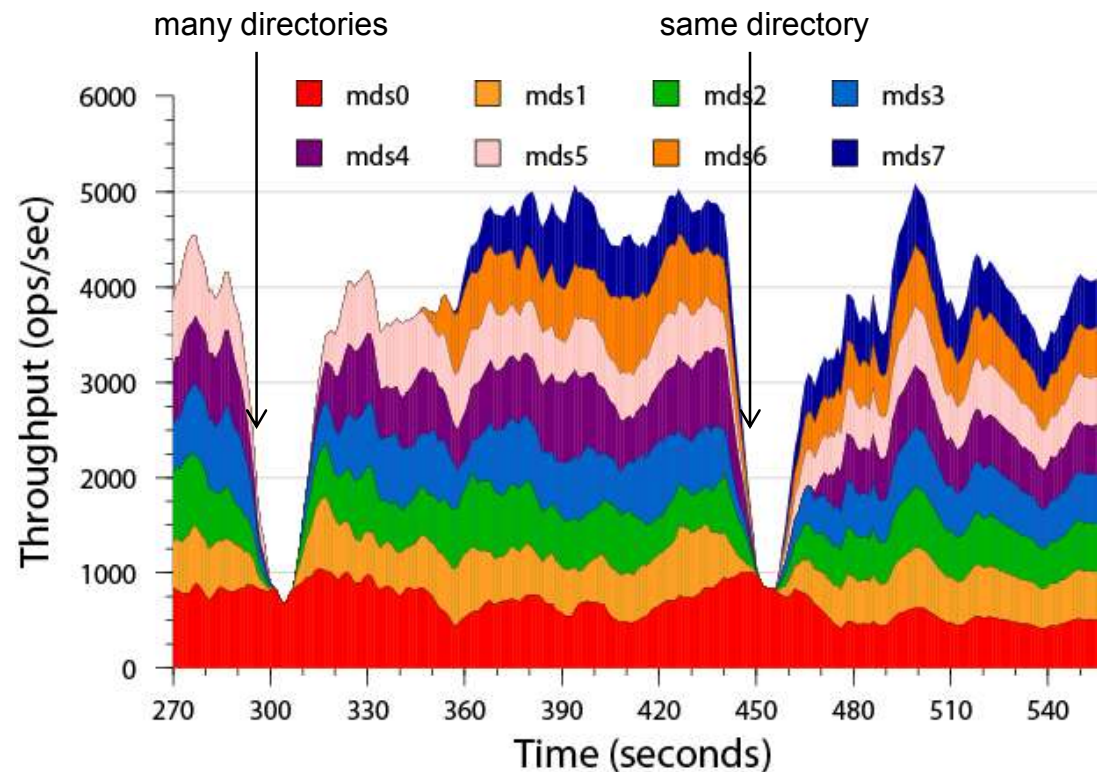
- Arbitrarily partition metadata, 10s-100s of nodes

□ Adaptive

- Move work from busy to idle servers
- Replicate popular metadata on multiple nodes

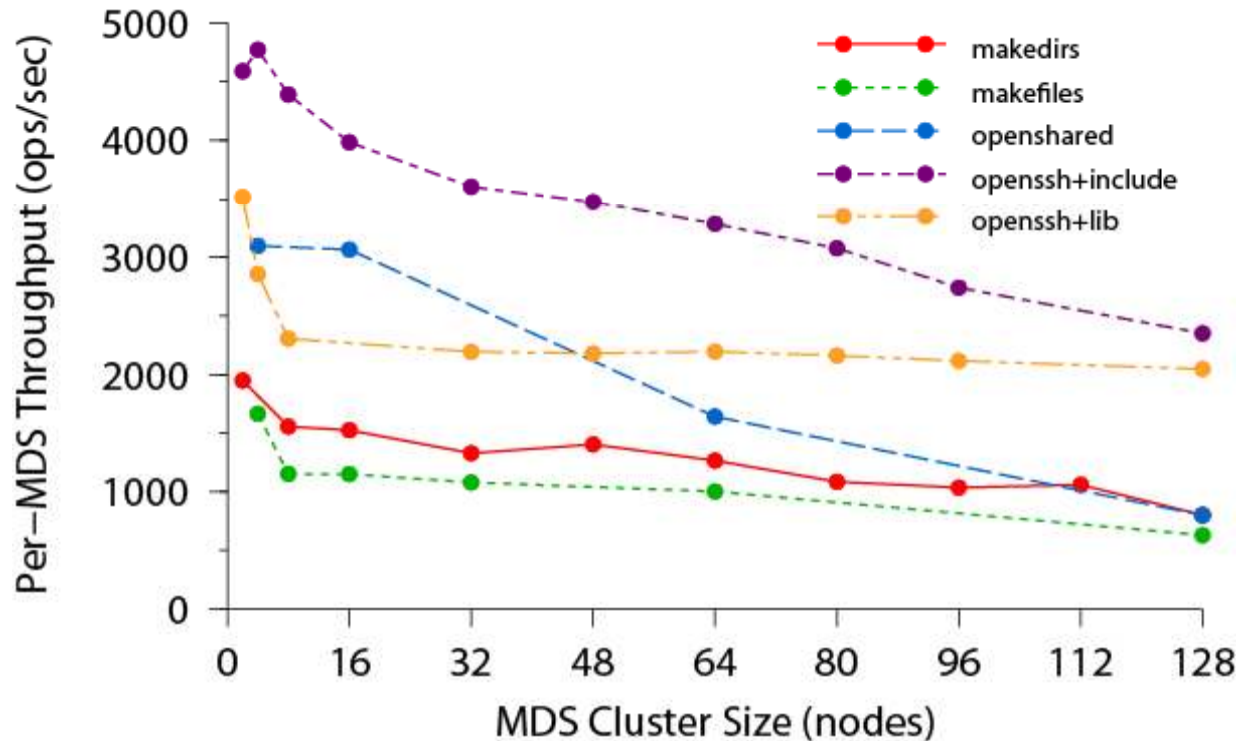
Workload adaptation

- Extreme shifts in workload result in redistribution of metadata across cluster
- Metadata initially managed by mds0 is migrated



Metadata scaling

- Up to 128 MDS nodes, and 250,000 metadata ops/second
- I/O rates of potentially many terabytes/second
- File systems containing many petabytes of data



Recursive accounting

□ Subtree-based usage accounting

□ Recursive file, directory, byte counts, mtime

```
$ 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
$ getfattr -d -m ceph. pomceph
# file: pomceph
ceph.dir.entries="39"
ceph.dir.files="37"
ceph.dir.rbytes="10550153946827"
ceph.dir.rctime="1298565125.590930000"
ceph.dir.rentries="2454401"
ceph.dir.rfiles="1585288"
ceph.dir.rsubdirs="869113"
ceph.dir.subdirs="2"
```

Fine-grained snapshots

- ❑ Snapshot arbitrary directory subtrees
 - ❑ Volume or subvolume granularity cumbersome at petabyte scale
- ❑ Simple interface

```
$ 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 foo/.snap/one
myfile bar/
$ rmdir foo/.snap/one # remove snapshot
```

- ❑ Efficient storage
 - ❑ Leverages copy-on-write at storage layer (btrfs)

File system clients

- ❑ POSIX; strong consistency semantics
 - ❑ Processes on different hosts interact as if on same host
 - ❑ Client maintains consistent data/metadata caches
- ❑ Linux kernel client
- ❑ Userspace client
 - ```
modprobe ceph
```

```
mount -t ceph 10.3.14.95:/mnt/ceph
```
  - ❑ **cfuse** FUSE-based client
  - ❑ libceph library (ceph\_open(), etc.)
  - ❑ Samba VFS, NFS-ganesha FSAL (libceph)
  - ❑ Hadoop FileSystem shim (libceph)

# Current status

- ❑ Linux client upstream since 2.6.34
- ❑ RBD client upstream since 2.6.37
- ❑ RBD client in Qemu/KVM and libvirt
- ❑ Debian, Fedora, Ubuntu packages
  
- ❑ Focus on stability
  - ❑ Bottom → top of stack
  - ❑ Object storage, RBD, single MDS, clustered MDS

# How is it built?

- C++
  - STL, boost
- C
  - Kernel code
  - CRUSH hashing
- Python
  - QA harness (teuthology)

- ❑ Teuthology = study of cephalopods
- ❑ Python testing infrastructure
  - ❑ Spin up cluster based on simple YAML description
    - ❑ # nodes, types, client types
  - ❑ Execute set of tasks, workloads
  - ❑ Clean up (and check for problems)
  - ❑ Node reservation/locking
  - ❑ Plays nice with valgrind, gitbuilder, gcov, etc.

# More information

- ❑ <http://ceph.newdream.net/>
  - ❑ Project wiki, tracker, news, mailing list
  
- ❑ Team is expanding
  - ❑ Kernel devs
  - ❑ C++ devs
  - ❑ QA, performance engineers
  - ❑ Downtown LA, Brea, San Francisco



# Open toolchain

- ❑ gcc/g++
- ❑ tcmalloc (google-perftools)
- ❑ crypto++
  - ❑ also libnss for RH benefit, but it sucks
- ❑ oprofile
- ❑ Coverity, cppcheck
- ❑ gitbuilder



