

GRAU DATA

Scalable OpenSource Storage

CEPH, LIO, OPENARCHIVE

RISINGTIDE
systems



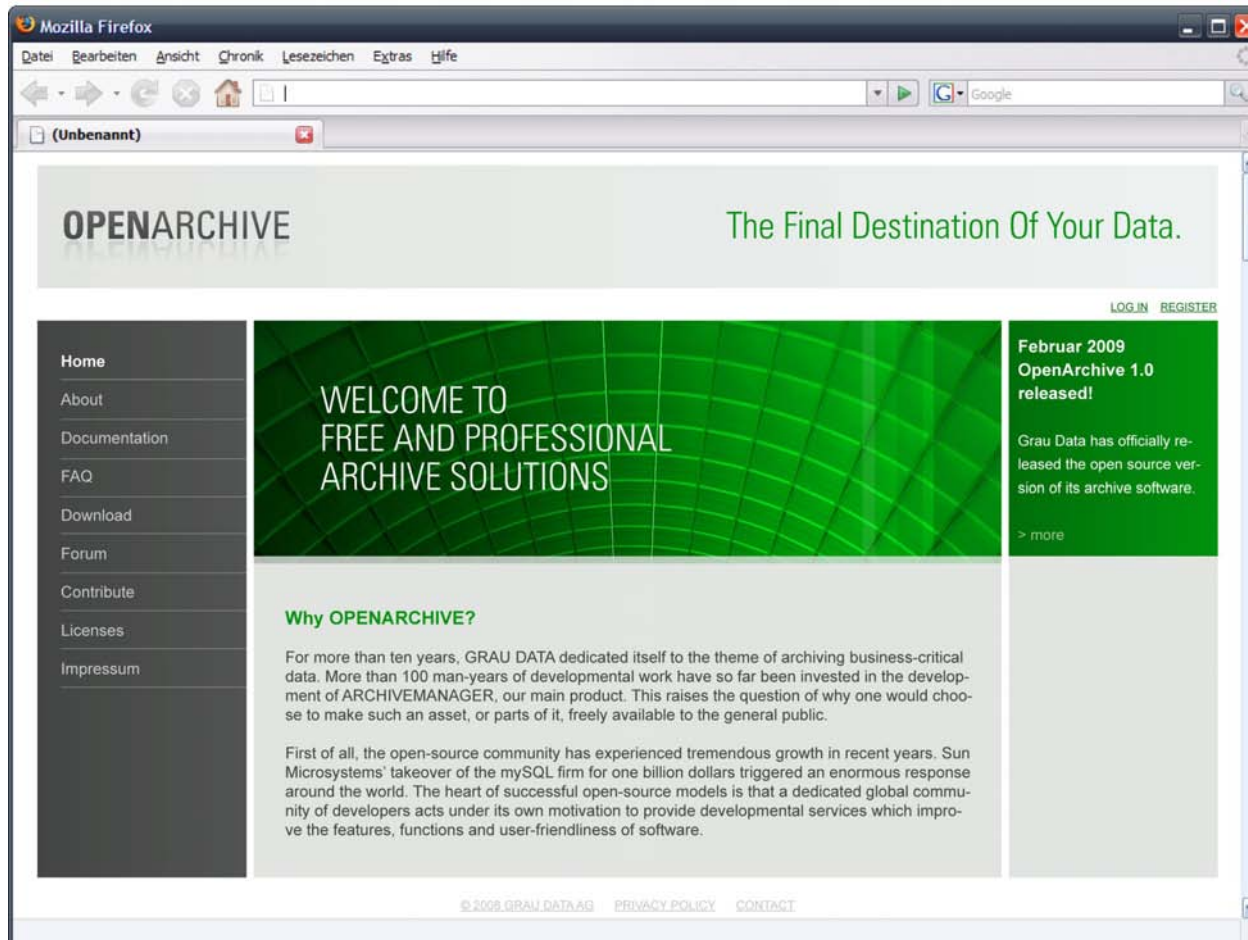
GRAU DATA

GRAU DATA: More than 20 years experience in storage



Mainframe Tape Libraries → Open System Tape Libraries
→ Archive Systeme → FMA Software
→ HP OEM FSE & FMA → **ARCHIVEMANAGER** → **OPENARCHIVE**

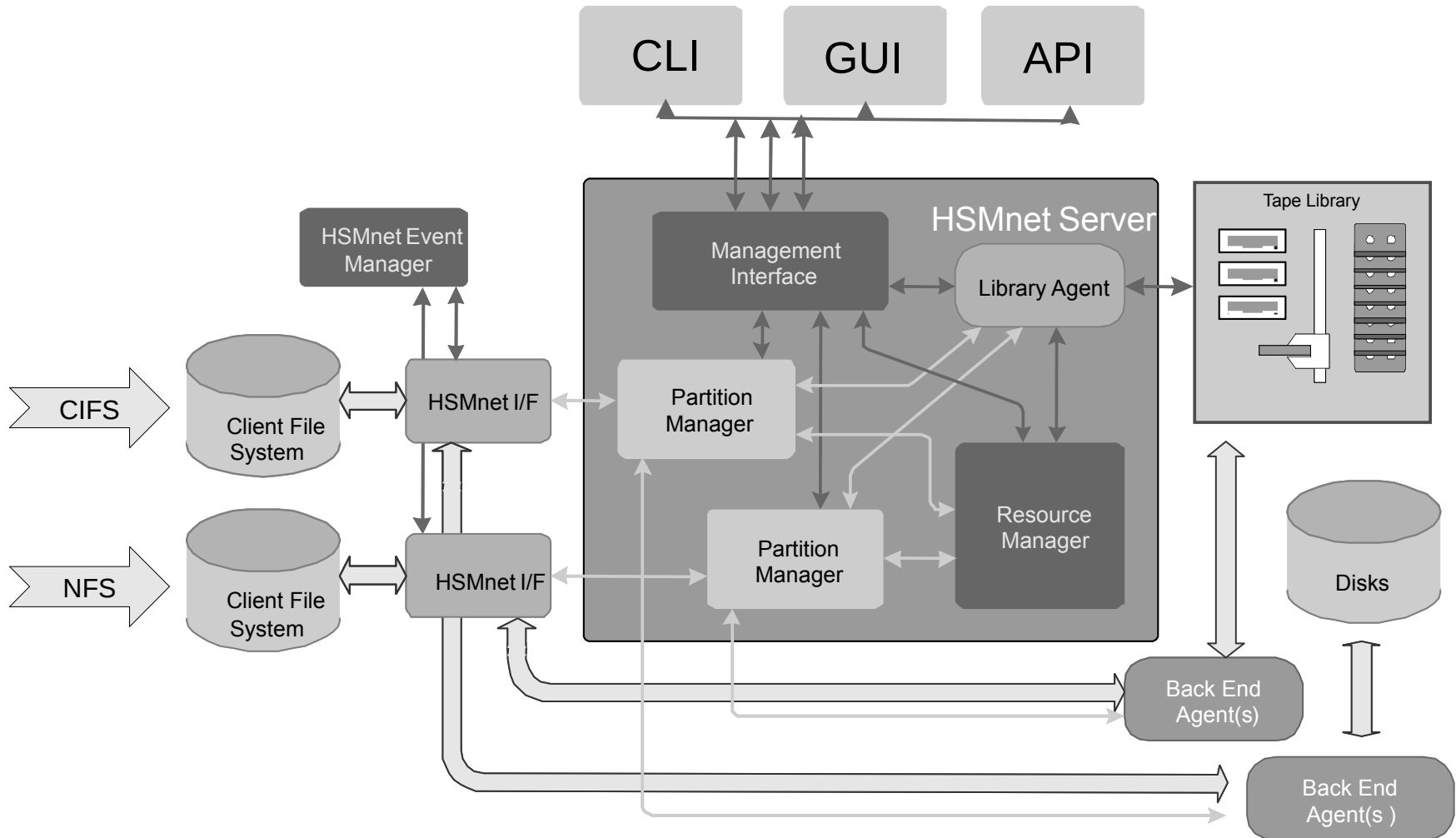
OPENARCHIVE - Homepage



OPENARCHIVE: 2nd generation of archiving software

- Start of development in 2000
- 100 man-years of development
- Scalable archive solution from 1 TB up to several PB
- Common code base for Windows and Linux
- HSM approach – data gets migrated to tapes or disks
- Filesystem interface (NTFS, POSIX) simplifies integration
- Support for all kinds of SCSI backend devices (FC, iSCSI)
- API only necessary for special purposes
- > 150 Customers worldwide
- References: Bundesarchiv, Charite, many DMS vendors

OPENARCHIVE: internal architecture



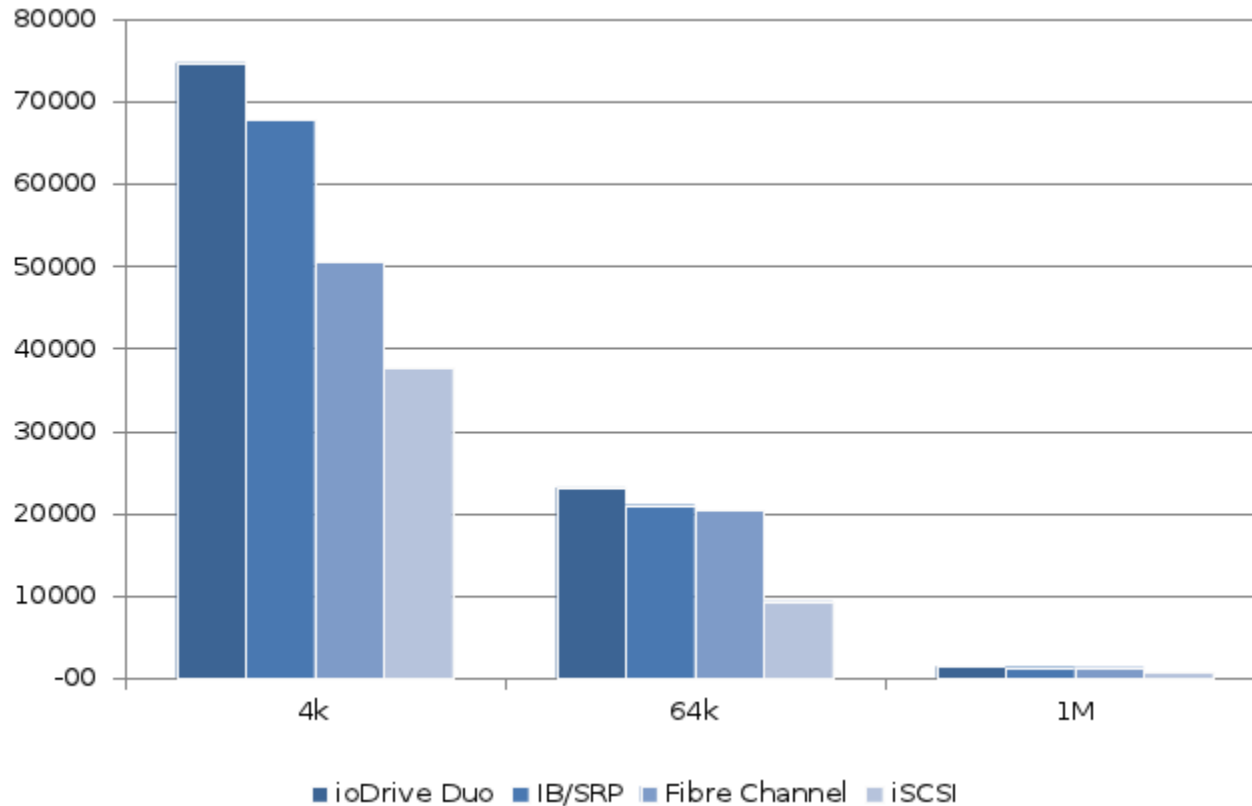
Current limitations of OPENARCHIVE

- Number of files and performance is limited by native filesystems (ext3/ext4, NTFS)
- Archiving and recalling of many files in parallel is slow
- Cluster file systems are not supported
- Performance is not appropriate for HPC environments
- Scalability might not be appropriate for cloud environments

High performance SCSI Target: LIO

- LIO is the new standard SCSI target in Linux since 2.6.38
- LIO is developed and supported by RisingTide System
- Completely kernel based SCSI target engine
- Support for cluster systems (PR, ALUA)
- Fabric modules for:
 - iSCSI (TCP/IP)
 - FCoE
 - FC (Qlogic)
 - Infiniband (SRP)
- Transparent blocklevel caching for SSD
- High speed iSCSI initiator (MP for performance and HA)
- RTSadmin for easy administration

LIO performance comparison (IOPS)



Based on 2 processes: 25% read – 75% write

iSCSI numbers with standard Debian client. RTS iSCSI Initiator plus kernel patches give much higher results.

Next generation ClusterFS: 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
- Developed by Sage Weil (Dreamhost) based on Ph.D. Theses
 - Upstream in Linux since 2.6.34 (client), 2.6.37 (RDB)

Ceph design goals

- Avoid traditional system designs
 - Single server bottlenecks, points of failure
 - Symmetric shared disk (SAN)
- Avoid manual workload partition
 - Data sets, usage grow over time
 - Data migration is tedious
- Avoid "passive" storage devices
 - Storage servers have CPUs; use them

Object storage

■ Objects

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

■ Object pools

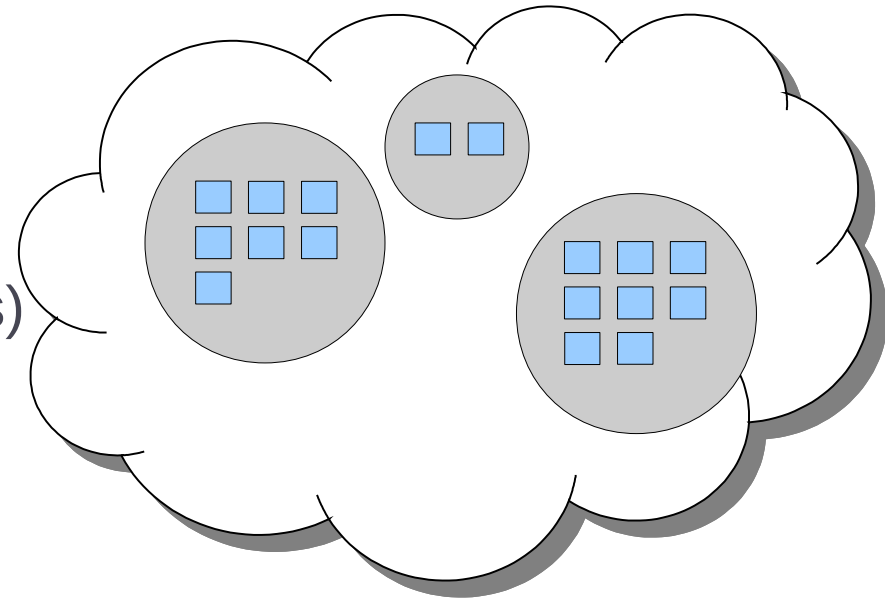
- Separate flat namespace

■ Cluster of servers store all objects

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

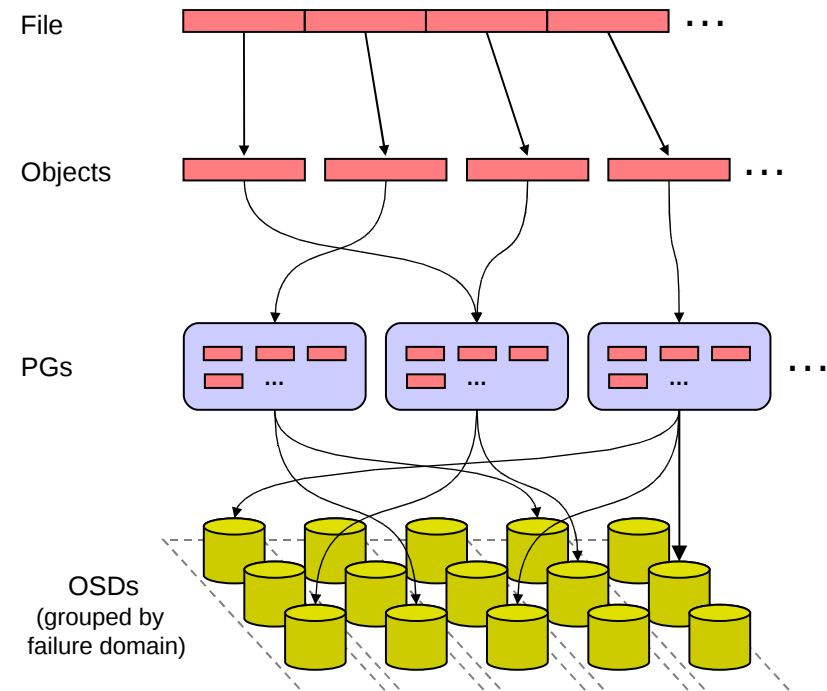
■ Low-level storage infrastructure

- librados, radosgw



Ceph data placement

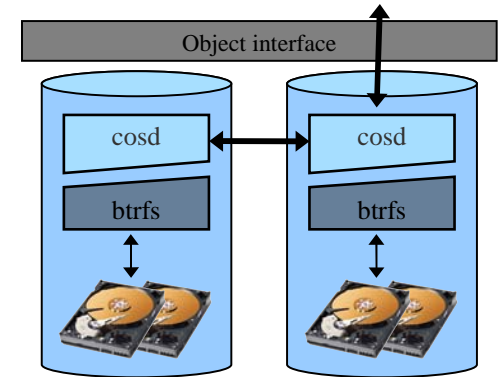
- Files striped over objects
 - 4 MB objects by default
- Objects mapped to *placement groups* (PGs)
 - $\text{pgid} = \text{hash}(\text{object}) \ \& \ \text{mask}$
- PGs mapped to sets of OSDs
 - $\text{crush}(\text{cluster}, \text{rule}, \text{pgid}) = [\text{osd2}, \text{osd3}]$
 - ~100 PGs per node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



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

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



It's all about object placement

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

Why btrfs?

- Featureful
 - Copy on write, snapshots, checksumming, multi-device
- 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] can also work...
 - Inefficient snapshots, journaling

Object storage interfaces: librados

- Direct, parallel access to entire OSD cluster
- PaaS, SaaS applications
 - When objects are more appropriate than files
- C, C++, Python, Ruby, Java, PHP bindings

```
rados_pool_t pool;

rados_connect(...);
rados_open_pool("mydata", &pool);

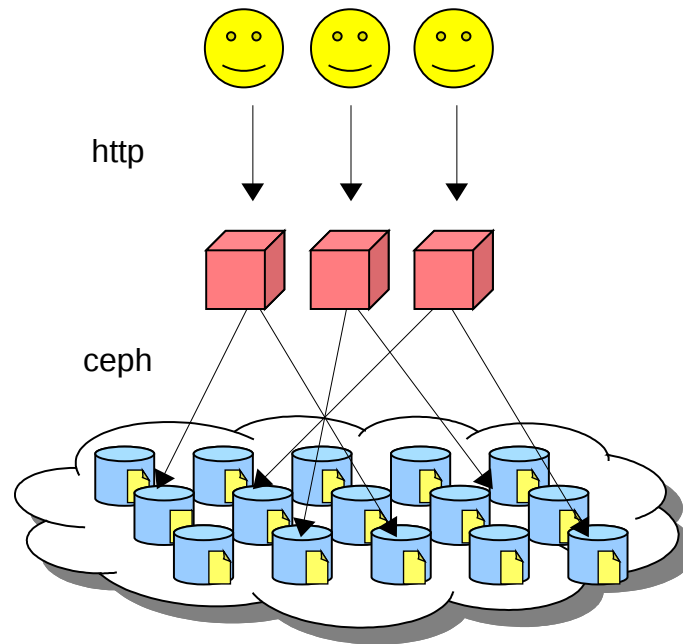
rados_write(pool, "foo", 0, buf1, buflen);
rados_read(pool, "bar", 0, buf2, buflen);
rados_exec(pool, "baz", "class", "method",
           inbuf, inlen, outbuf, outlen);

rados_snap_create(pool, "newsnap");
rados_set_snap(pool, "oldsnap");
rados_read(pool, "bar", 0, buf2, buflen); /* old! */

rados_close_pool(pool);
rados_deinitialize();
```

Object storage interfaces: radosgw

- Proxy: no direct client access to storage nodes
- HTTP RESTful gateway
 - S3 and Swift protocols



RDB: RADOS block device

- Virtual disk image striped over objects
 - Reliable shared storage
 - Centralized management
 - VM migration between hosts
- Thinly provisioned
 - Consume disk only when image is written to
- Per-image snapshots
- Layering (WIP)
 - Copy-on-write overlay over existing 'gold' image
 - Fast creation or migration

RDB: RADOS block device

- Native Qemu/KVM (and libvirt) support

```
$ 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
```

- Simple administration

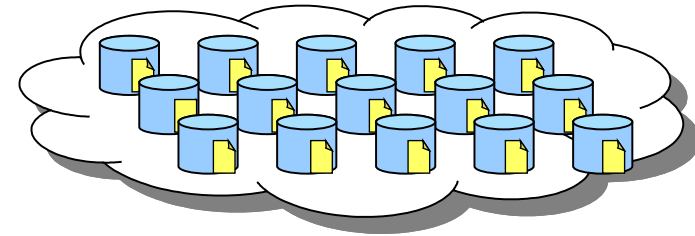
- CLI, librbd

```
$ 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
```

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", 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

POSIX filesystem

- Create file system hierarchy 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 dynamically
 - Automagically load balanced

POSIX example

- `fd=open("/foo/bar", O_RDONLY)`

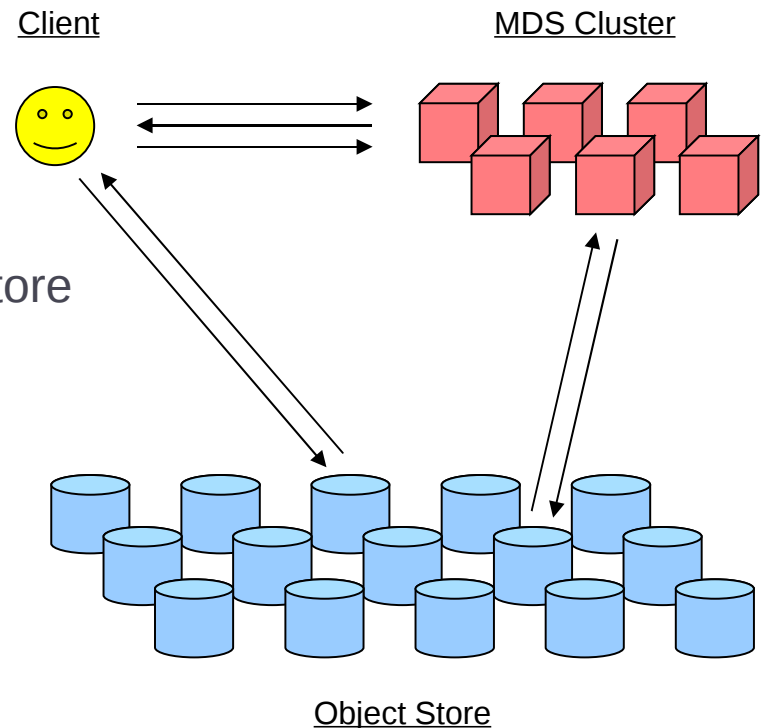
- Client: requests open from MDS
- MDS: reads directory /foo from object store
- MDS: issues capability for file content

- `read(fd, buf, 1024)`

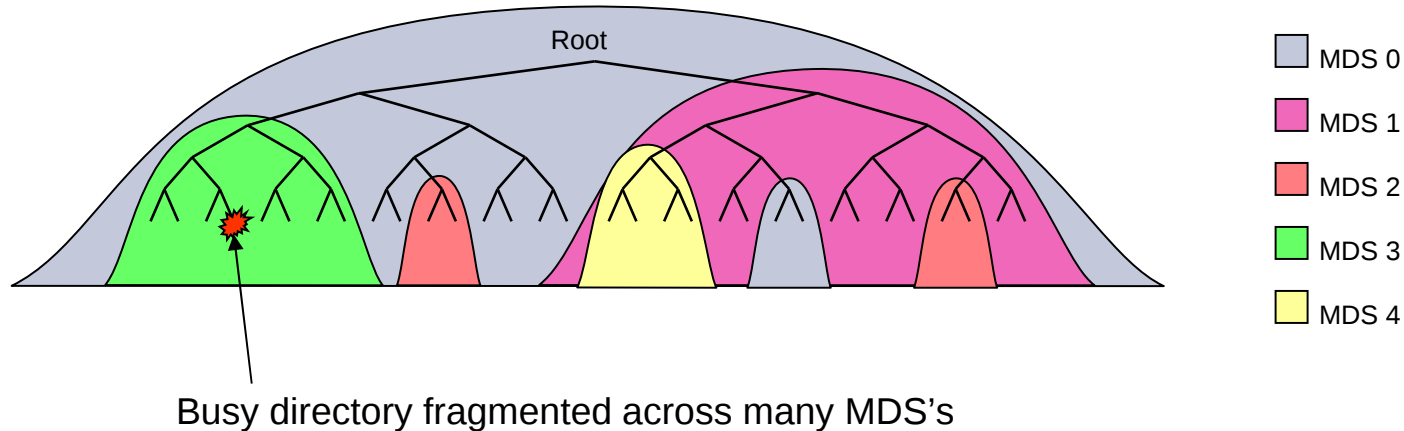
- Client: reads data from object store

- `close(fd)`

- Client: relinquishes capability to MDS
- MDS out of I/O path
- Object locations are well known—calculated from object name



Dynamic subtree partitioning



■ Scalable

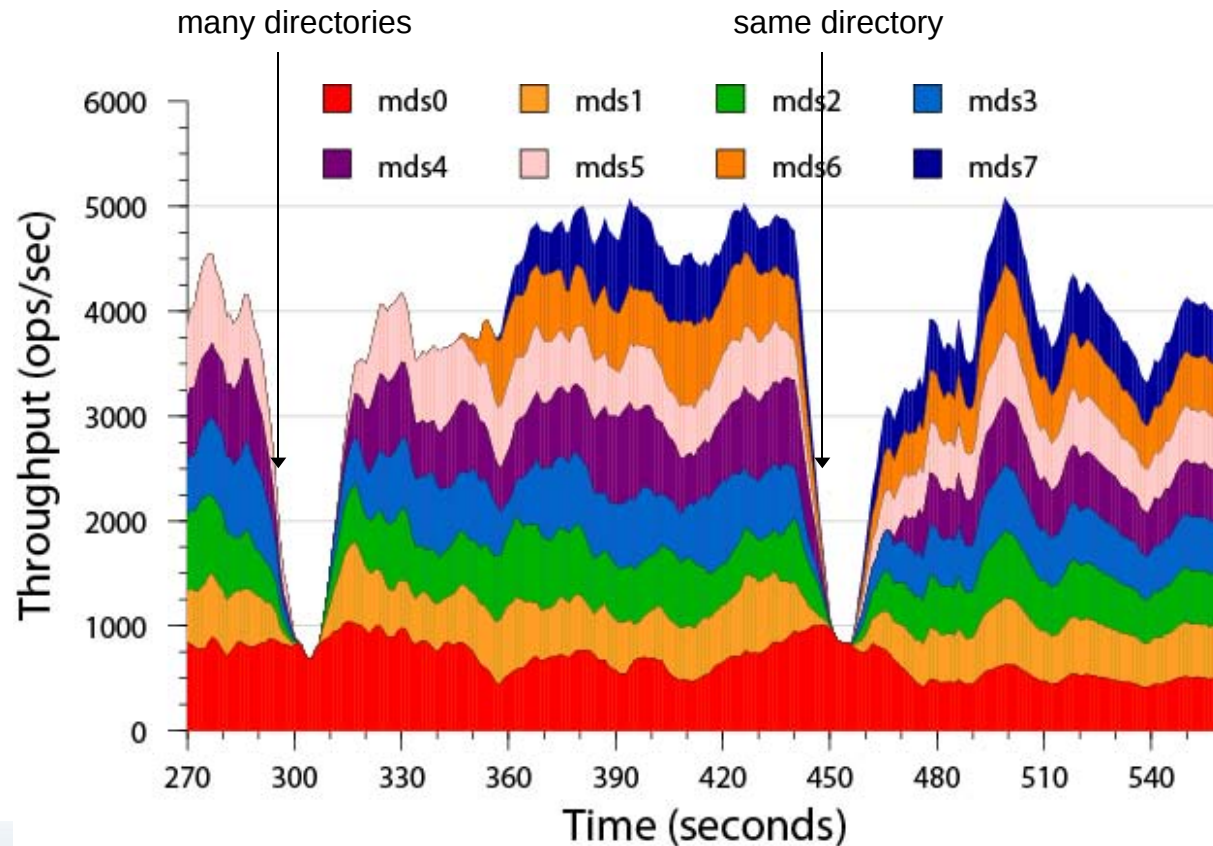
- Arbitrarily partition metadata, 10s-100s of nodes

■ Adaptive

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

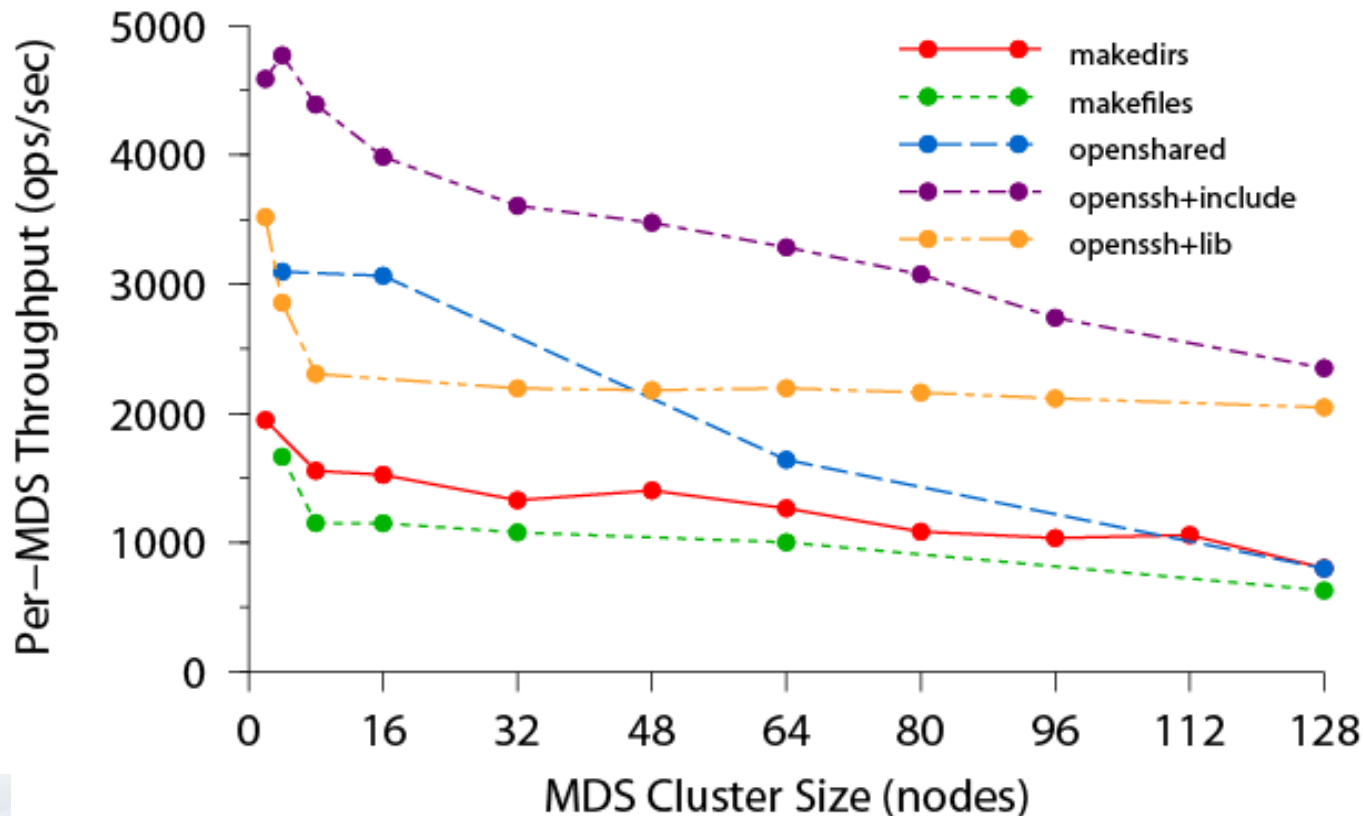
Workload adaptation

- 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)

POSIX filesystem client

- POSIX; strong consistency semantics
 - Processes on different hosts interact as if on same host
 - Client maintains consistent data/metadata caches

- Linux kernel client

```
# modprobe ceph
# mount -t ceph 10.3.14.95:/ /mnt/ceph
# df -h /mnt/ceph
```

Filesystem	Size	Used	Avail	Use%	Mounted on
	10.3.14.95:/	95T	29T	66T 31%	/mnt/ceph

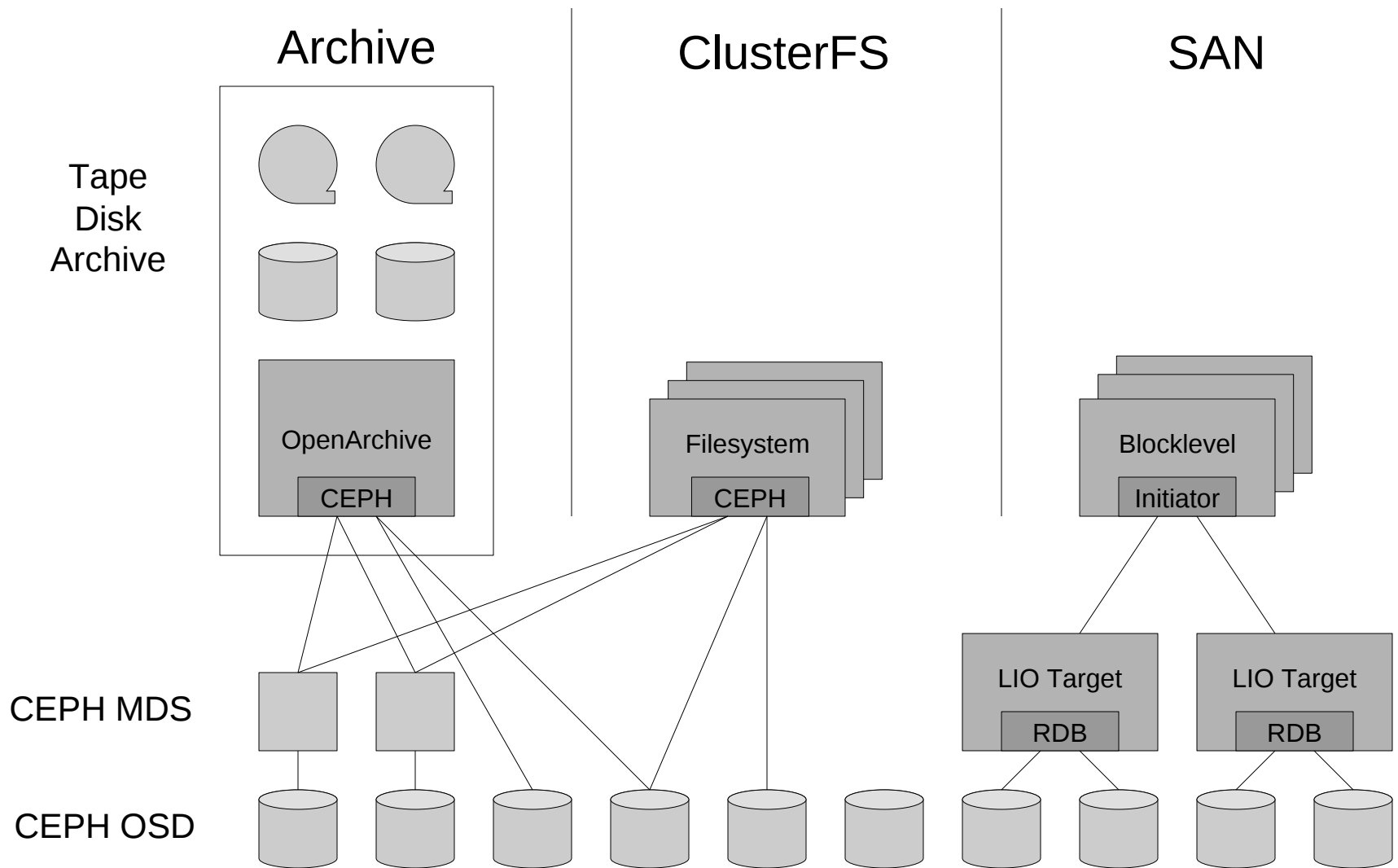
- Userspace client

- **ceph-fuse** FUSE-based client
- libceph library (ceph_open(), etc.)

Integration of CEPH, LIO, OpenArchive

- CEPH implements complete POSIX semantics
- OpenArchive plugs into VFS layer of Linux
- Integrated setup of CEPH, LIO and OpenArchive provides:
 - Scalable cluster filesystem
 - Scalable HSM for cloud and HPC setups
 - Scalable high available SAN storage (iSCSI, FC, IB)

I have a dream...



Contact

Thomas Uhl

Cell: +49 170 7917711

thomas.uhl@graudata.com

tlu@risingtidesystems.com

www.twitter.com/tuhl

de.wikipedia.org/wiki/Thomas_Uhl