whamcloud

European Lustre Workshop
Paris, France
September 2011

ORIon Project

Alex Zhuravlev, Mike Pershin
Lustre Engineers
Whamcloud, Inc.

What ORIon project is

Stands for OSD Restructuring Initiative

Provide a single storage abstraction
OSD API introduced in Lustre 2.0
OSD API only used for MDS metadata operations
Redundant APIs left for OSS, MDS object ops, llog
Facilitate different backend storage systems

OSD API and Services

OSD API becomes rich
Enough to implement OSS, MDS, MGS
All components use OSD API:
MDD, OFD, MGS, llog, Changelogs
Old APIs can be removed to simplify code
Well defined MDS stack
OSD Proxy (OSP) uses OSD API to interface to OSTs
OSP isolates network RPCs from MDD layer
Simplifies error-prone code

OSD API: Benefits

Easier to exploit new backend storage system
ZFS well underway today
Btrfs discussed for the future, when stable
Able to interface with non-filesystem backends?
Easier code
Semantically clear object API
Modules resolve specific problems internally
Less efforts to become Lustre developer
More development from the community

OSD API: changes in details

2-stage transactions:
- Declare, execute
- Inspired by ZFS
- Allows to get rid of magical credits in the code
- Stackable

Methods to manipulate data:
- 0-copy IO
- Punch (truncate)
- Caching is hidden by specific OSD

Commit callbacks
- Per transaction

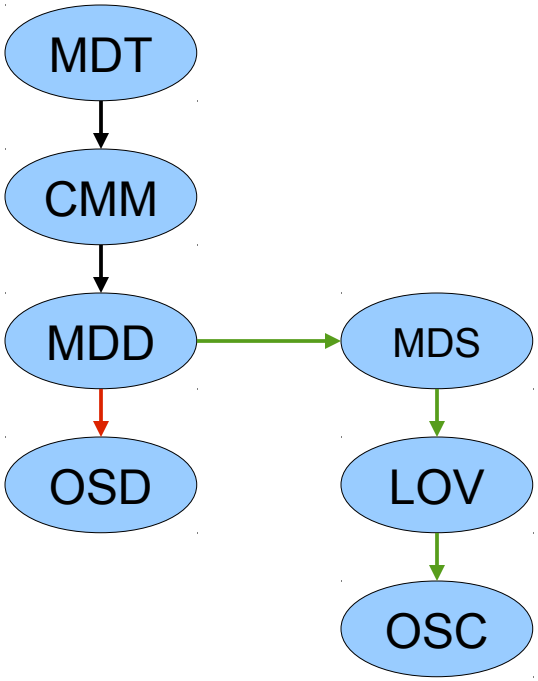
Infrastructure

Changes to facilitate different disk filesystems:
Userspace utilities
Mkfs.lustre, mount.lustre
Tests and testing infrastructure
Do not depend on ldiskfs specifics when possible
Implement equivalent tests where needed

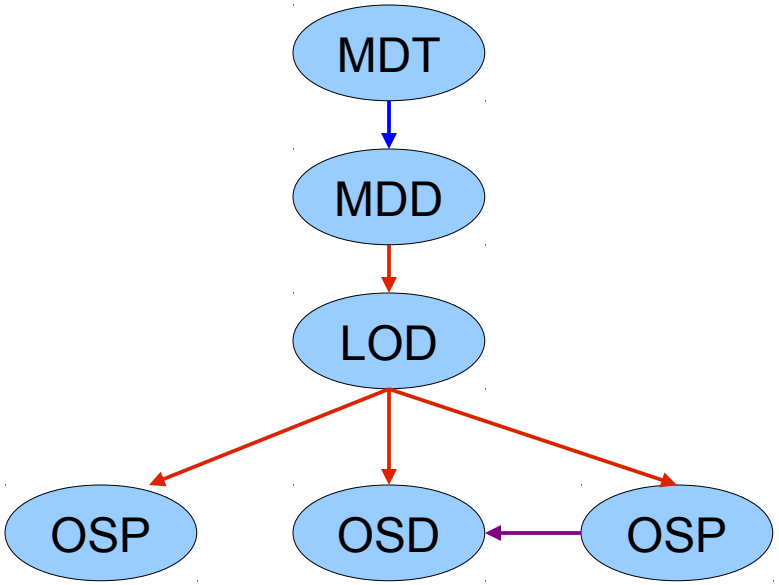
OST objects are destroyed by MDS

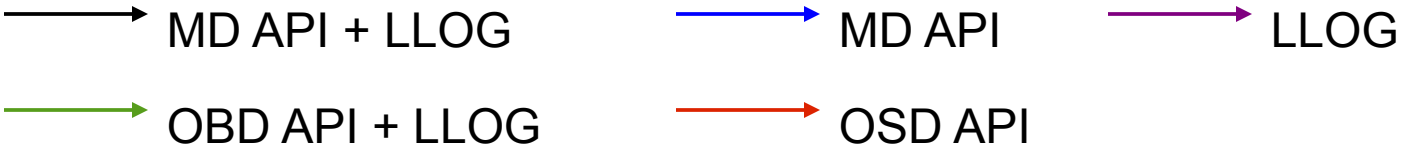
In 1.8/2.0 destroy sent by the clients to OSTs
Lots of plumbing needed for distributed transactions
Vulnerable to double failures
Can result in file without objects after some failures
No data loss (user really deleted file), but annoying
In ORIon destroy sent by MDS to OSTs
Really atomic (commit on MDS first)
In batches
Explores OSD API for distributed operations
 Next step is DNE Phase 1

2.0/2.1



2.X





Create logic (mdt_reint_create)

```
mdd_create
  create tx
    lod_declare_object_create
      osd_declare_object_create
      osp_declare_object_create
        precreate/reserve object, declare local changes
    lod_index_declare_insert
      osd_index_declare_insert
  start tx
    lod_object_create
      osd_object_create
      osp_object_create
        assign object id, store last id on local store
    lod_index_insert
      osd_index_insert
  stop tx
```

File creation, hidden part

mdt_reint_create() and the whole path down are synchronous

We still need async precreates

Dedicated thread in OSP is introduced
    Specifically to do precreations in background

Sync internally
    Easy to follow, maintain

Unlink logic (mdt_reint_unlink)

```
mdd_unlink
  create tx
    lod_declare_object_destroy
      osd_declare_object_destroy
      osp_declare_object_destroy
        declare llog record
    lod_declare_delete
      osd_declare_delete
  start tx
    lod_object_destroy
      osd_object_destroy
      osp_object_destroy
        write llog record
    lod_index_delete
      osd_index_delete
  stop tx
```

File unlink, hidden part

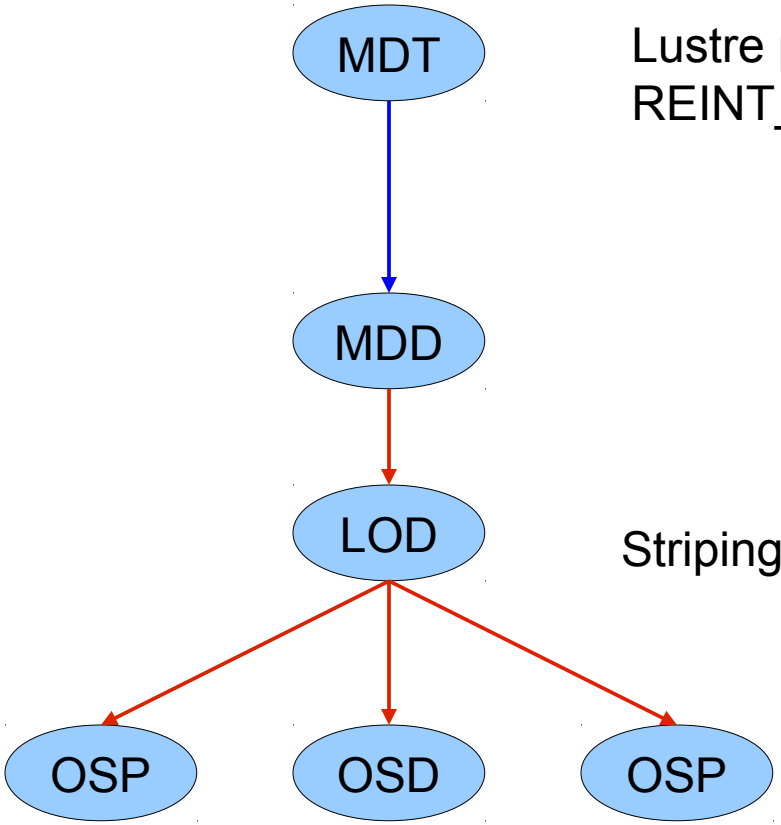
mdt_reint_unlink () and the whole path down are synchronous
    Just stores llog records in atomic manner

Another dedicated thread in OSP is introduced
    Specifically to handle OST object removal

Sync internally
    Easy to follow, maintain

Will be doing in batches

2.X model of MDS stack



Lustre protocol. Complex operations (RPCs) from file c
REINT_OPEN := create + open + getattr + getxattr(LO

Posix file operations from primitives (updates):
Create := obj create + insert

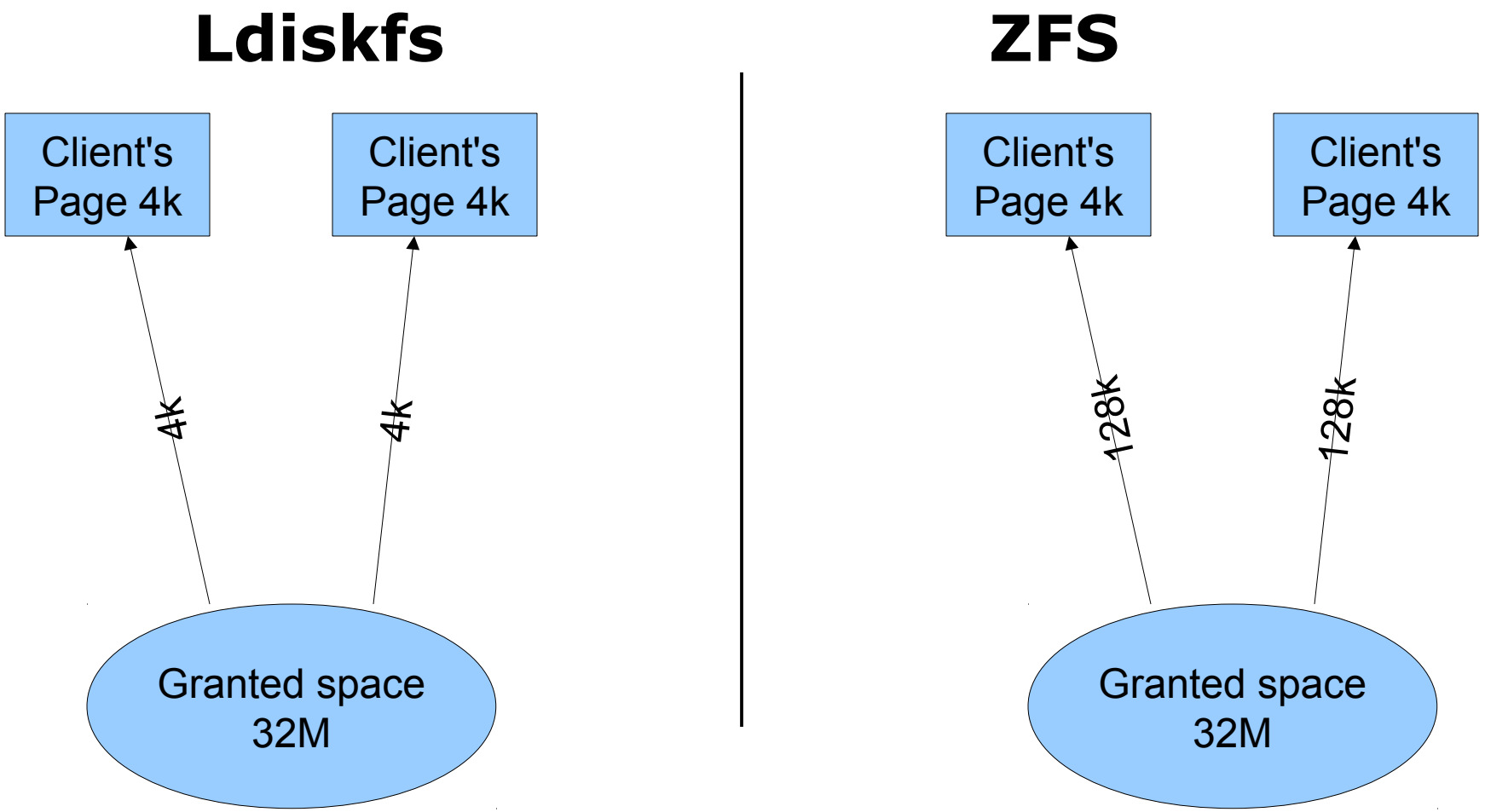
Striping, access to striped objects, updates within trans

Access to single object,
updates within transaction
OSD – local objs, OSP – remote objs

Changes to the grants subsystem

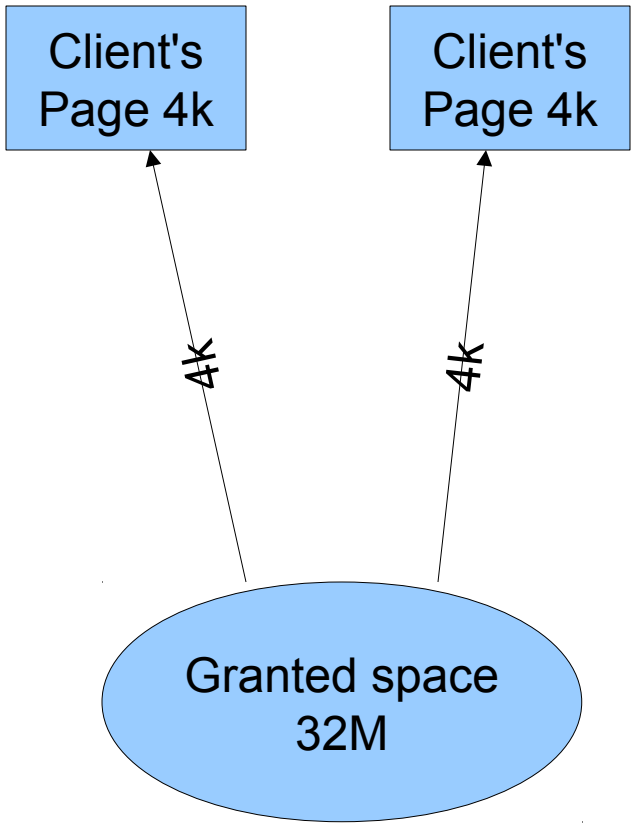
Grants enable client data writeback caches
Ensure sufficient space remains to clean dirty cache
1.8/2.0 implementation too ldiskfs specific
Supports 4K blocksize only
ZFS, btrfs, ext4 (soon) support much larger block sizes
ORIon introduces a grant block size
Contiguous pages in the same write RPC share block charge

1.8/2.0 schema

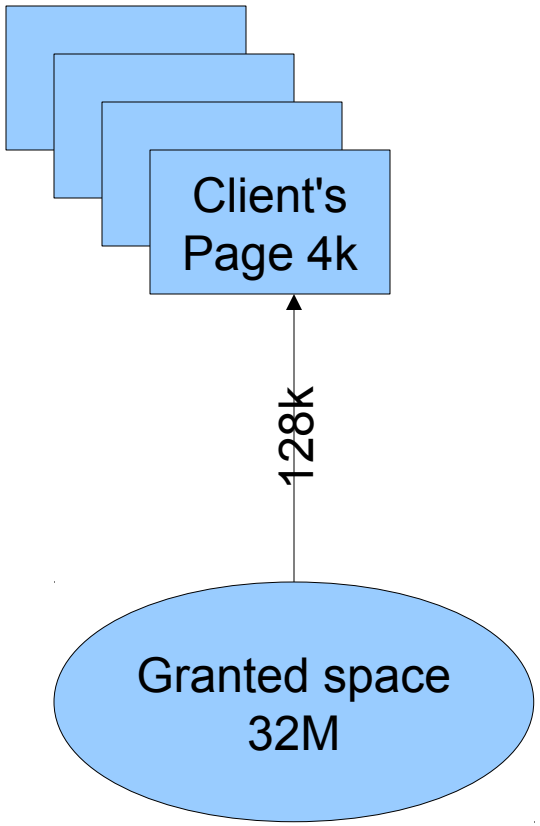


ORIon schema

Ldiskfs



ZFS



1.8/2.0 quota is not portable

Relies on Linux generic quota mechanism
part of VFS
We can't reuse that with ZFS
ZFS does not support this framework
OSD API bypasses VFS
VFS does not have transaction support
VFS constrains IO to be in PAGE_SIZE chunks, inefficient
VFS has unscalable locking (only one lock per file/directory)
Restructuring
Quota *accounting* in OSD
Backend-specific metadata overhead details
Quota *enforcement* in Lustre
Generic, based on accounting data from OSD

Quota becomes

Portable
Accounting is part of OSD disk-specific code
Access quota information via OSD API
Flexible
Enforcement in Lustre
Can balance precision vs. performance
Scalable
Quota info requested directly from OSS
Clients previously requested all quota info via MDS

Thank You

Alex Zhuravlev, Mike Pershin
Lustre Engineers
Whamcloud, Inc.