Lustre Quotas

SC09, Portland, Nov 2009

Johann Lombardi
Lustre Group
Sun Microsystems

Topics

- Architecture Overview (9 slides)
- Adaptive qunit
- Performance challenge & CMD support
- Quotas on DMU
- Quota overruns & interaction with client's caches
- Quota recovery review & concerns

Topics

- Architecture Overview (9 slides)
- Adaptive qunit
- Performance challenge & CMD support
- Quotas on DMU
- Quota overruns & interaction with client's caches
- Quota recovery review & concerns

Architecture Primer

- A centralized server hold the cluster wide limits: the quota master(s)
  - guarantees that global quota limits are not exceeded
  - track quota usage on slaves
- Quota slaves
  - all the OSTs and MDT(s)
  - manage local quota usage/hardlimit
  - acquire/release quota space from the master

Quota Master(s)

- 1.4/1.6/2.0: 1 single master running on the MDS
- In charge of:
  - storing the quota limits for each uid/gid
  - accounting how much quota space has been granted to slaves
- quota information are stored in administrative quota files
  - files proper to Lustre (admin_quotafile.usr/grp)
  - format identical to the one used in the VFS
  - Has mistakenly been using ->write/read for a long time
    - Fixed in 1.8.2 to use journaled operations (read_record/write_record)

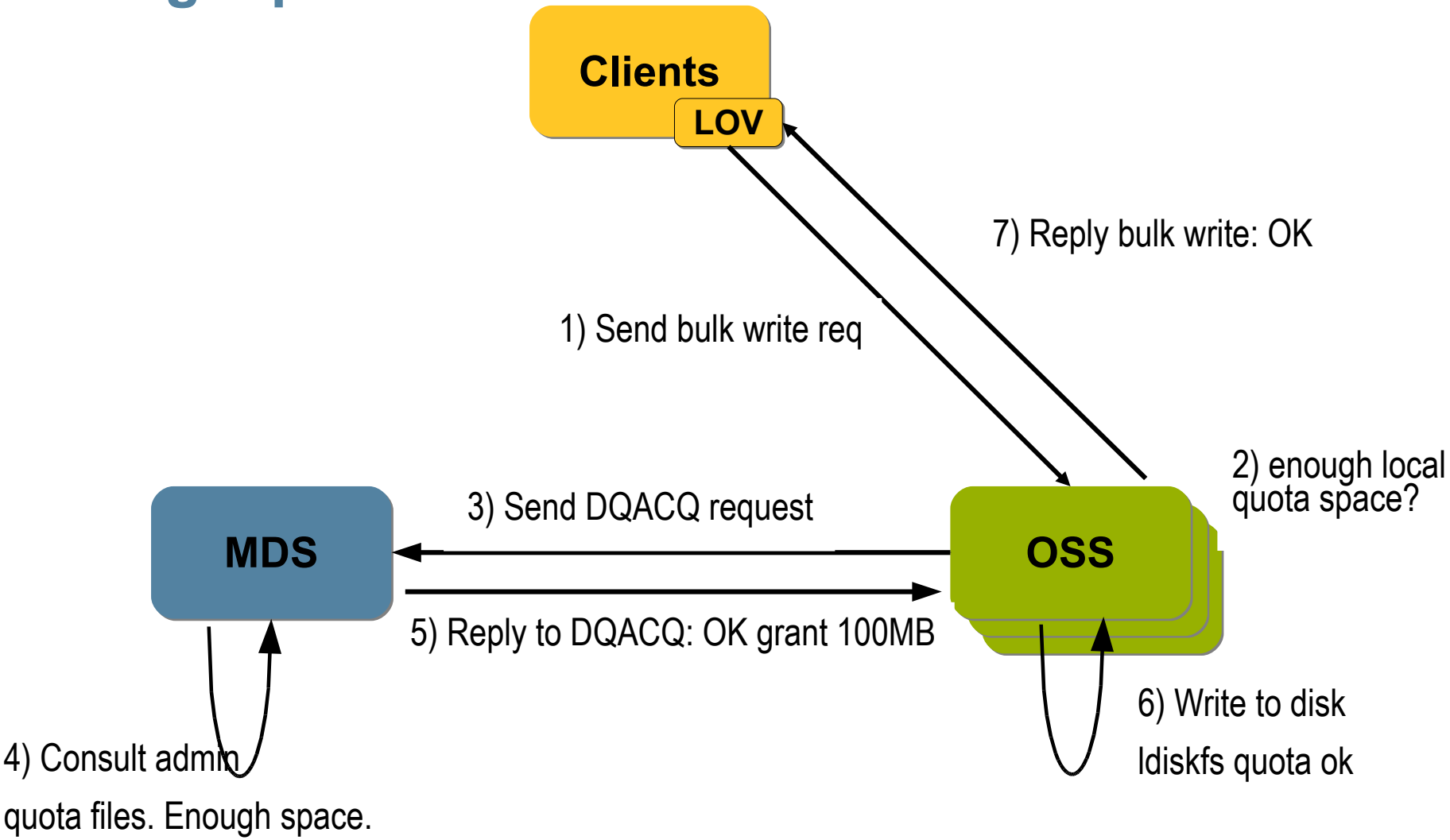
Quotas Slaves

- All OSTs and MDT(s)
- Rely on ldiskfs quotas
  - only use hard limit, not soft limit
  - operational quota files are managed by ldiskfs (journaled quotas since 1.6.5)
  - accounting is handled by ldiskfs too
- In charge of returning EDQUOT (quota exceeded) to the clients when quota is exhausted

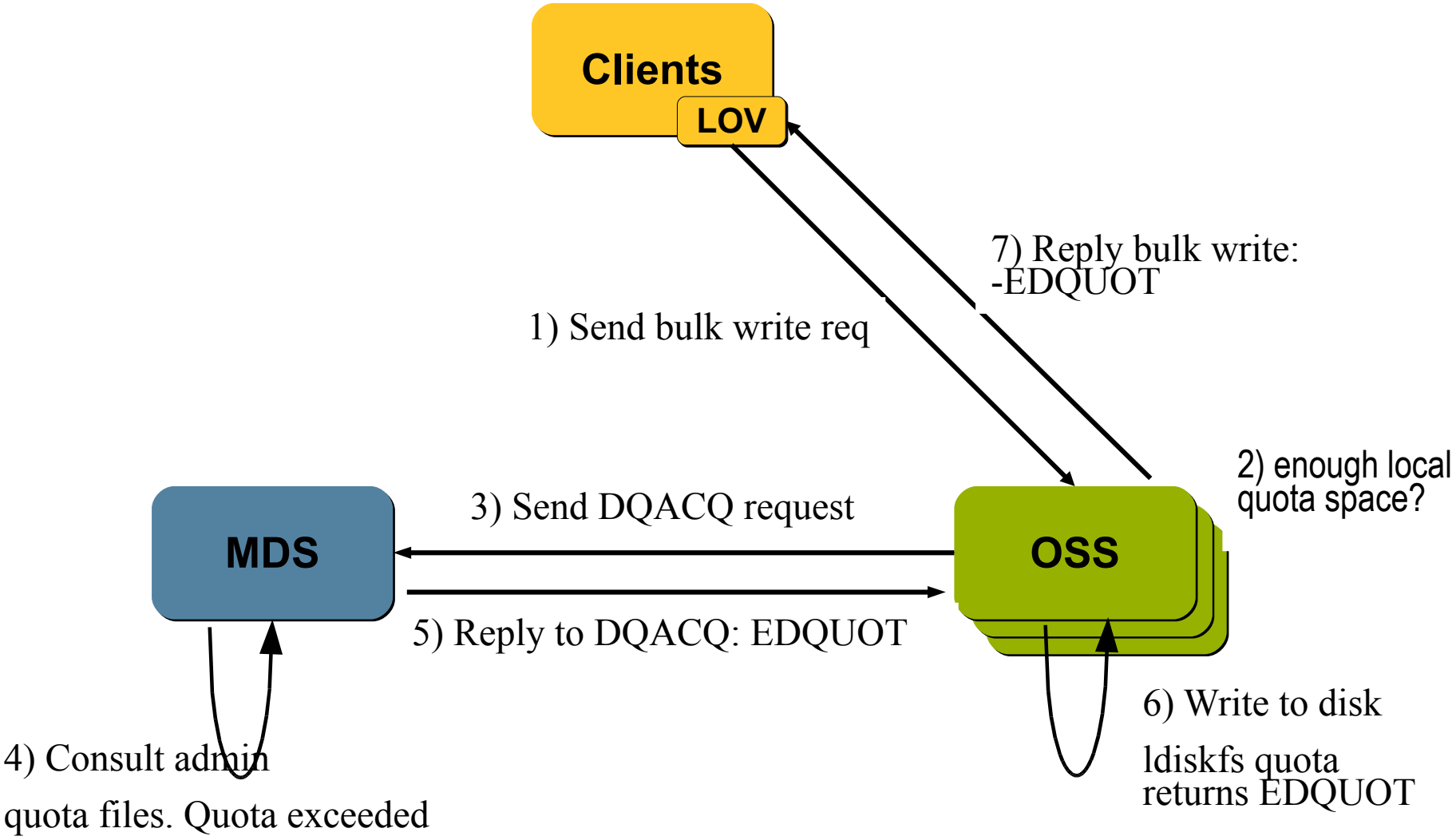
Acquire/Release Protocol

- Two different RPC types
  - DQACQ = Disk Quota ACQuire
  - DQREL = Disk Quota RELease
- DQACQ/DQREL RPCs are
  - initiated by slaves
  - processed by master(s)
- increase/lower the local **hardlimit** on slaves
- increase/decrease administrative **usage** on the master

Quota protocol overview:
Enough quota



Quota protocol overview:
Quota exceeded - EDQUOT



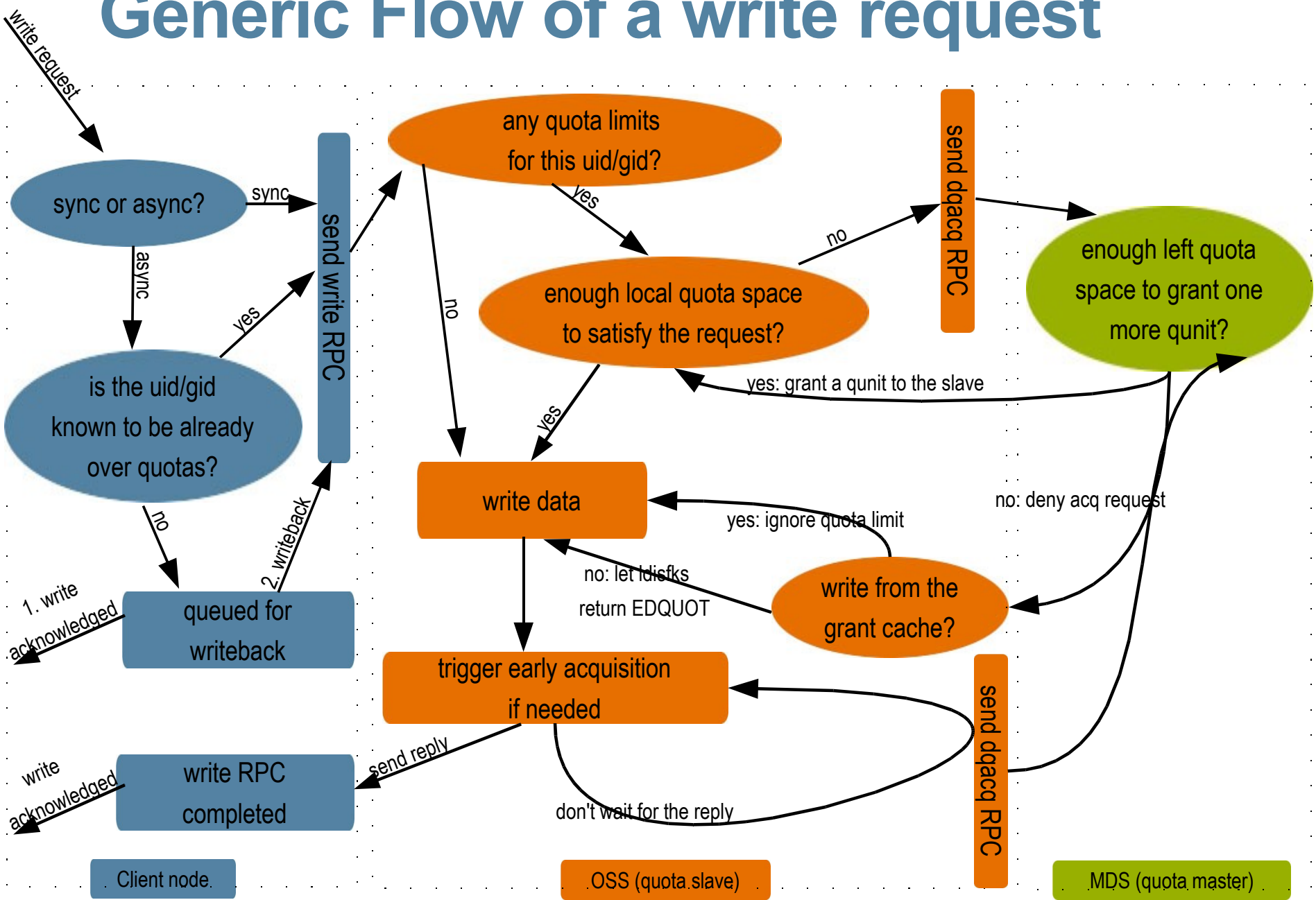
Quota space acquisition

- For performance reasons, quota slaves don't acquire quota for each write request
- The master grants quota to slaves by blocks of qunit
  - iunit/bunit default value 5120/128MB
  - will need to be bumped soon for performance
  - if many writes underway, slaves can try to acquire more
- early qunit acquisition to improve performance
  - Slaves also proactively acquire qunit ahead of time
  - If remaining quota space < qtune
    - a DQACQ RPC is sent
  - If remaining quota space > qtune + qunit
    - a DQREL RPC is sent

Quota flow on the slaves

- Estimate space needed to handle the request
  - Also take into account metadata blocks
    - extent tree depth
    - If not accounted, we may not acquire enough, causing ldiskfs to return spurious EDQUOT
- If active writes + current usage < local hardlimit
  - Don't acquire more and let ldiskfs handle the write
- Otherwise
  - Acquire space from master
  - Write request could be stuck for a long time if master not ready
    - Fixed in bug 20530
- At most one quota rpc in flight for a given uid/gid

Generic Flow of a write request



Topics

- Architecture Overview
- Adaptive qunit
- Performance challenge & CMD support
- Quotas on DMU
- Quota overruns & interaction with client's caches
- Quota recovery review & concerns

Problem with static qunit value

- Slaves can have up to qunit + qtune of unused quota space
- Quota space granted by the master cannot be claimed back
- Consequence:
  - If the master has already granted all the quota space to slaves, some slaves may return EDQUOT while some others still have free quota space

150MB quota free
=> can still handle

150MB quota free
=> can still handle

0MB quota free & MDS has no more quota space
=> Return EDQUOT

User perception

- What happens from the user point of view:
  - writes on objects stored on OST3 returns EDQUOT
  - 'lfs quota' reports quota usage far from limit
  - writes on objects stored on OST1 & 2 are successful
- Users/Admins expect quotas to work on lustre like on any local fs and are disturbed by this

150MB quota free
=> can still handle

150MB quota free
=> can still handle

0MB quota free & MDS has no more quota space
=> Return EDQUOT

Adapting qunit value dynamically

- The idea is quite simple:
  - enlarge qunit size when far from quota limit
  - shrink qunit size when getting closer to quota limit
- The dynamic qunit patch improves
  - quota accuracy when close to quota limit
    - the new qunit size is broadcasted to slaves after shrinking
  - support for small quotas
  - allow us to bump qunit significantly
    - needed for performance
    - without leaking too much quota space
- Landed for 1.4.12 and 1.6.5
  - bug 10600

Qunit shrink/enlarge policy

- **quota_boundary_factor:** thresholds triggering qunit inc/decrease
  - If left_quota < quota_boundary_factor * ost_num * current_qunit,
    - qunit is shrunk
  - If left_quota > 2* quota_boundary_factor * ost_num * current_qunit
    - qunit is bumped
  - default value of quota_boundary_factor is 4
- Factor by which qunit size grows/shrinks: quota_qs_factor
  - Default value is 2
- min/max qunit value can be set: **quota_least_qunit** / **quota_qunit**
  - default set to 1/5120 for inodes and 1MB/128MB for blocks
- If quota usage is oscillating around the threshold, we don't want to change qunit too often
  - **quota_switch_seconds** is how long to wait before growing again after shrinking
  - default value is 300s

Informing slaves of qunit change

- As said, unused quota space cannot be claimed back …
- But the new qunit value is broadcasted to all the slaves
  - opc OST_QUOTA_ADJUST_QUNIT sent by master to slaves
  - Inform slaves of new qunit value
  - Slave releases unused quota space according to new value
  - Not sent in parallel today :(, patch under testing
- Address quota space leak issue mentioned before
  - Definitely improved accuracy
  - Still not a reliable solution
    - Master doesn't wait for slave to ack the broadcast before processing new dqacq req
    - Being addressed in bug 17381

Topics

- Architecture Overview
- Adaptive qunit
- Performance challenge & CMD support
- Quotas on DMU
- Quota overruns & interaction with client's caches
- Quota recovery review & concerns

Impact on Performance (1/2)

- Additional actions are required on slaves when quotas are enabled
  - ldiskfs needs to maintain block/inode accounting for each uid/gid
  - qunit must be acquired from the master
    - additional RPCs are required
- Enabling quotas has no significant performance impact today because
  - The early qunit acquisition algorithm looks pretty efficient
  - The quota master is powerful enough to handle quota requests in a timely manner
- We now have many quota statistics to investigate performance issue
  - bug 15058, landed in 1.6.6

Impact on Performance (2/2)

- Still, performance challenges remain
  - 2,000 OSTs @ 500MB/s with 100MB qunit requires 10,000 RPCs to be processed on the master
- Thoughts:
  - Using several quota masters
  - Increasing qunit (max qunit size is 128MB today)
  - Granting more to slaves initially and relying on the broadcast mechanism to claim unused qunits back
  - Improvement to the dynamic qunit are needed

CMD Support

- May want to use several quota master to spread the load across several MDSs
- Provide uid-gid / MDT mapping
  - Hash on uid/gid does not work well with dynamic MDT addition
- But one given uid is still limited to one master
  - Not a problem if we bump qunit & improve broadcast mechanism

Topics

- Architecture Overview
- Adaptive qunit
- Performance challenge & CMD support
- Quotas on DMU
- Quota overruns & interaction with client's caches
- Quota recovery review & concerns

DMU/ZFS quota

- Used to only support quotas on fileset
- per uid/gid quota support has been landed to ZFS
  - Need to migrate to a new “layout”
  - Quota accounting always enabled, no quotacheck functionality is provided
    - what if accounting goes wrong …
  - Quota not really accurate since we don't exactly know how much space will be needed
- DMU
  - register callback invoked when file is written to disk
    - ZPL registers its own callback
  - Provide API to consult current disk usage

Supporting quota on top of DMU

- Interfacing with DMU API
  - > Register our own callback
  - > Get current disk usage
- Estimating how much space needed for a write
  - > And returning EDQUOT from lquota instead of ldiskfs
- Maintaining our own operational quota files on slaves

Space Accounting with DMU

- Already have data structures storing per-uid info
  - aka lqs
  - records pending write, req in flight, current qunit size, ...
- Registering our own callback to DMU
  - Just update current usage & pending write when this callback is called
  - DMU updates accounting on disk as part of same transaction

What to account?

- difficult to predict how much space is needed
  - because of metadata blocks
- But less important than with ldiskfs
  - since quota exceeded is now returned by lquota
- Should just make a reasonable estimation
  - some quota overruns is tolerated
  - or just discard metadata blocks totally?

Storing quota info on disk

- Maintaining our own operational quota files on slaves
  - Should not be big deal since we already do this on master with administrative quota files
  - Just need to store hardlimit
  - Using sparse files? (Nikita)
    - Indexed by uid/gid

Some other things to think about ...

- Rewrite allocates new blocks
  - > Need to make sure it is accounted correctly
- Capability to ignore quota enforcement
- Porting to DMU requires to change the quota interface
  - > same scheme can be implemented with ldiskfs
  - > Do we want/have to do this?
    - – No, means supporting 2 different quotas APIs at the same time

Topics

- Architecture Overview
- Adaptive qunit
- Performance challenge & CMD support
- Quotas on DMU
- Quota overruns & interaction with client's caches
- Quota recovery review & concerns

Quota overruns

- Client nodes cache dirty data behind server's back
  - Up to max_dirty_mb (=32MB) per OSC
  - Grant cache prevents getting ENOSPC on writeback
- Today, no interactions between the grant cache and quotas
- If a user is over quota already, slaves
  - still accept writes from the grant cache
  - but inform the client in the reply that it should stop caching dirty data for this uid/gid
  - This causes quota overruns that can be **significant**
    - Worst case scenario: # clients * # ost * 32MB

Workaround landed ....

- Ask the client to stop caching data sooner rather than later
- Tunable via /proc, namely quota_sync_blk (bug16642)
- Unfortunately, does not address all the cases

How to address quota overruns?

- introducing some quota knowledge on the client
  - Allow granting quota space to client
- Quota space could be granted as part of DLM locks
  - Claim quota space back via callbacks (glimpses)
- Merging quota & grant space
  - although quota is per-user/group
  - We don't always know the uid/gid on lock enqueue
    - But can be fixed easily

Topics

- Architecture Overview
- Adaptive qunit
- Performance challenge & CMD support
- Quotas on DMU
- Quota overruns & interaction with client's caches
- Quota recovery review & concerns

Quota recovery

- Quotas info are now journaled on both master & slaves
- Master recovery
  - master contacts all slaves and asks for local hardlimit
  - compute global quota usage and update admin quota files
  - If one slave is missing, recovery is aborted
- Slave recovery
  - Check current usage against hardlimit
  - acq/rel unused quota space above/below qunit + qtune

Slave (re)integration

- OST addition (online or not) is not handled properly
- quotacheck needs to be run first on a new OST
  - but currently, this requires a full quotacheck :(
  - would be easy to fix by triggering quotacheck once the OST joins the fs
- Worse, the new OST is not said what users have quota enforced
  - so this new OST won't try to acquire space from master
- Same can happen if one OST has been down for some time
  - Won't see updates on quota limit
- Holes in slave recovery

Lustre quotas

Johann Lombardi
johann@sun.com