Lustre Quotas

HPC workshop, Germany, Sept 2009

Johann Lombardi
Lustre Group
Sun Microsystems

Topics

- Architecture Overview
- Shortcomings & solutions
- Performance impact
- Quotas on DMU

Initial Requirements

- Ability to enforce both block and inode quotas
- Hard and soft limits are supported
- Central utility to set/get limits/usage and initialize quotacheck operation
- No significant performance impact

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
- For CMD support
  - use several quota masters
  - use a hash on the uid/gid

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

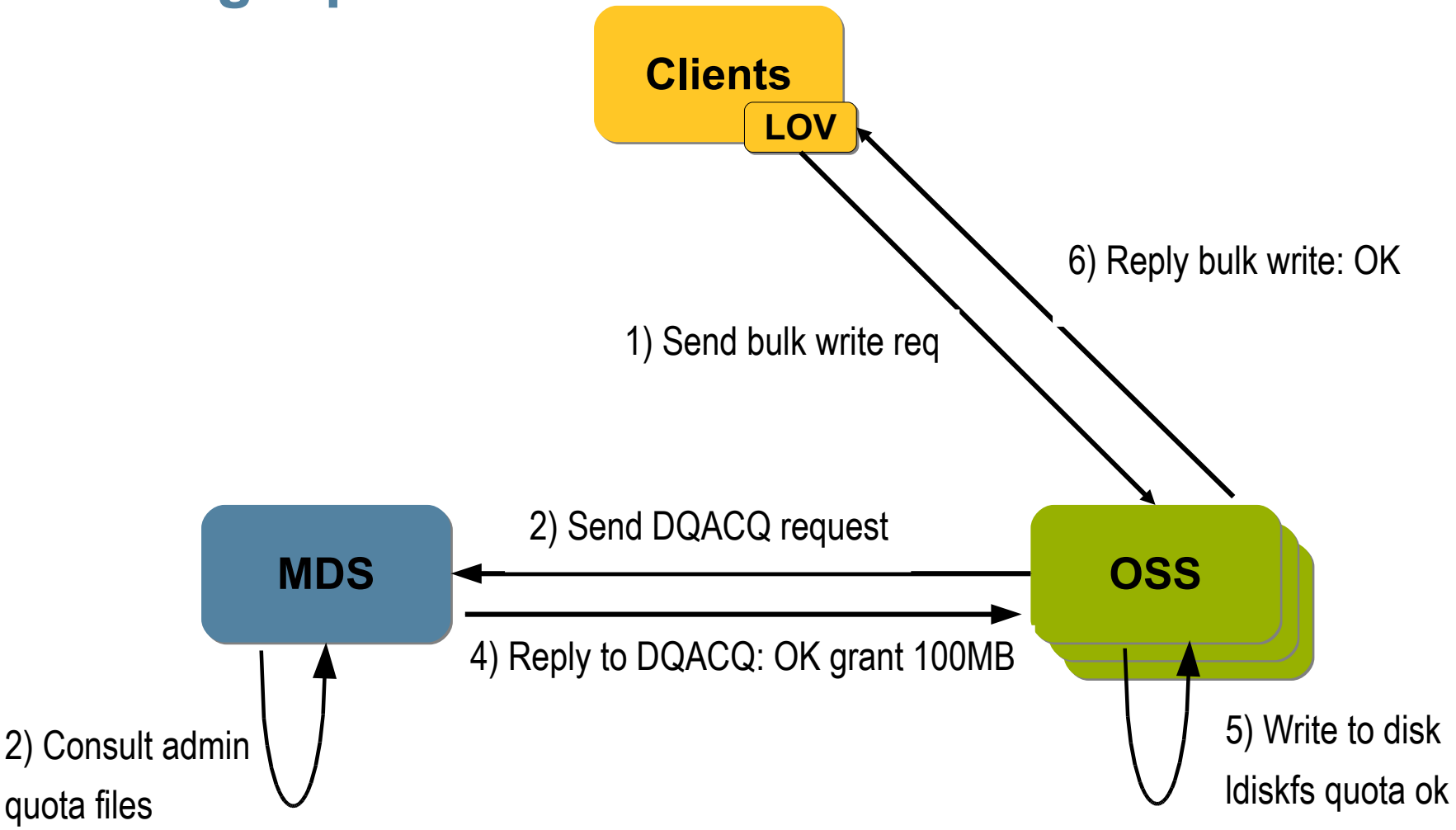
Acquire/Release Protocol

- 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

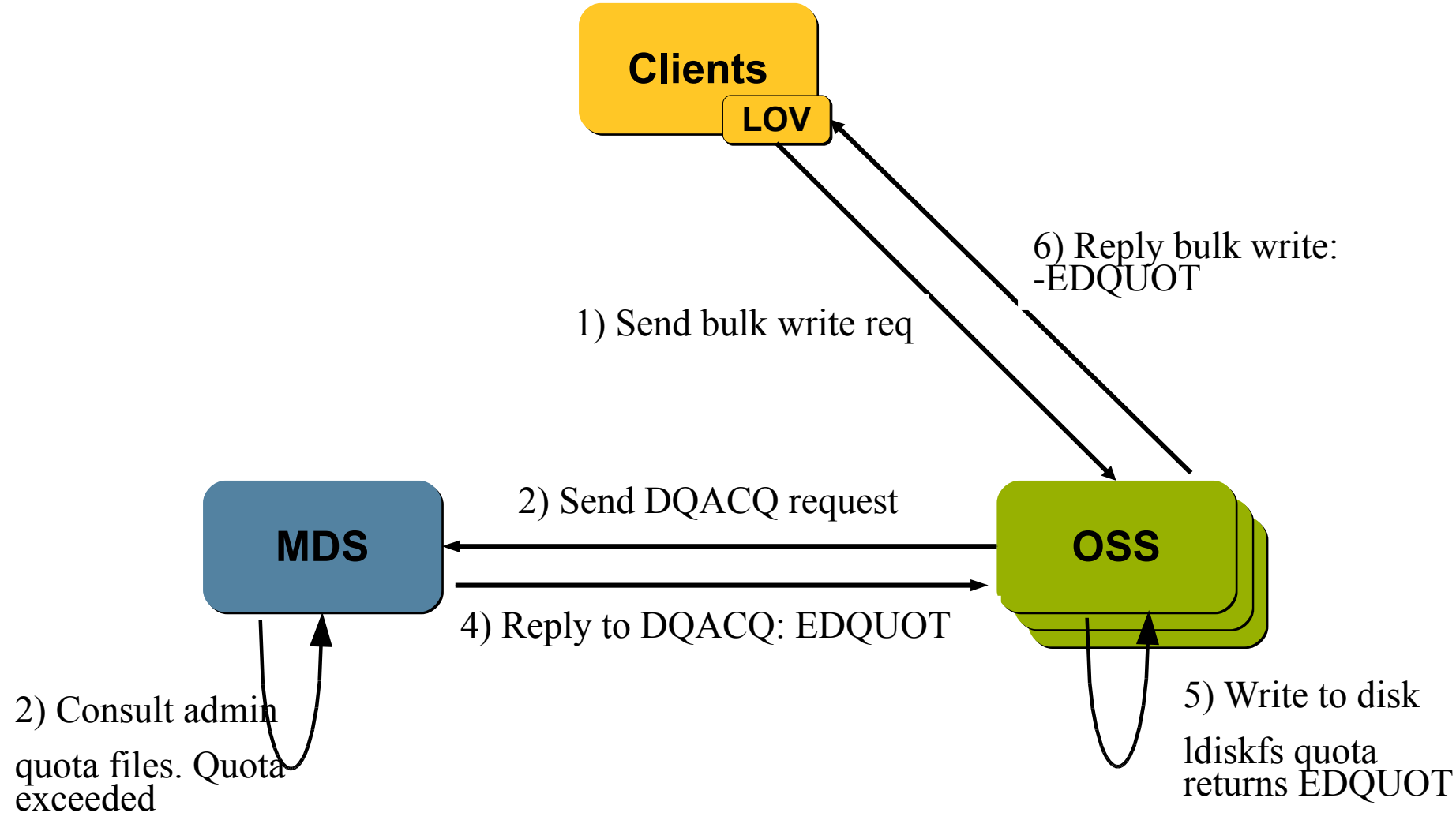
Running out of quota

- EDQUOT = quota exceeded
- quota slaves return this error when:
  - the remaining quota space is not sufficient to satisfy the write request
  - AND the master cannot grant additional quota space to the slave
- EDQUOT is returned by ldiskfs

Quota protocol overview:
Enough quota



Quota protocol overview:
Quota exceeded



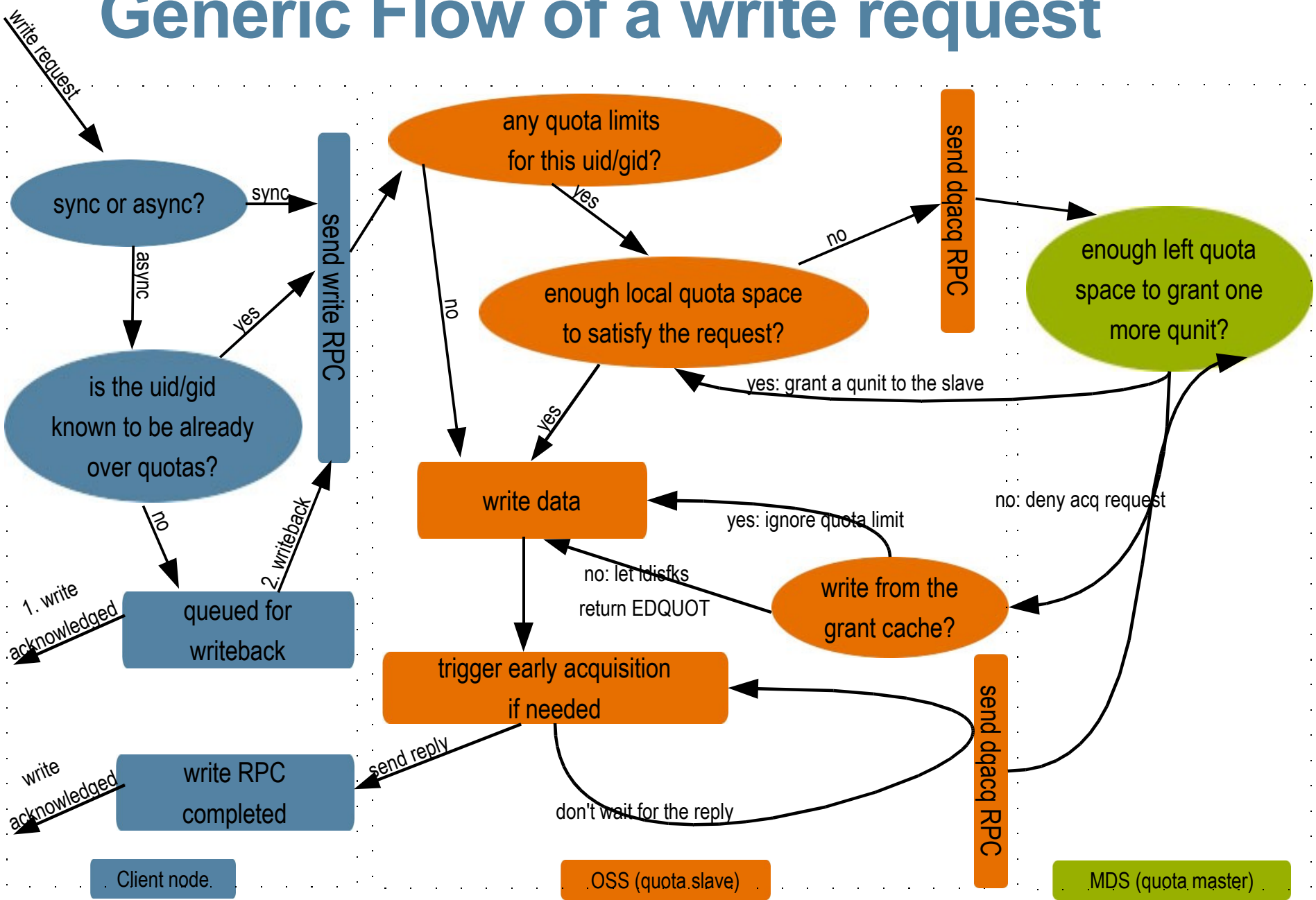
Quota space acquisition

- For performance reasons, quota slaves don't acquire quota for each write request
- The master grants quota to slaves by blocks of qunit
- iunit for inodes
  - default value is 5120 inodes
- bunit for blocks
  - default value is 128MB

Early qunit acquisition

- Slaves proactively acquire qunit ahead of time
  - early qunit acquisition to improve performance
- Once a request's been processed, slaves “adjust” the local limit
  - If remaining quota space < qtune
    - a DQACQ RPC is sent
  - If remaining quota space > qtune + qunit
    - a DQREL RPC is sent
- itune for inodes / btune for blocks
  - default value is ½ qunit

Generic Flow of a write request



Issue #1: Space leak

- 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 writes

150MB quota free
=> can still handle writes

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

Issue #1: Space leak

- What happens from the user point of view:
  - > Writes on objects stored on OST3 returns EDQUOT while 'lfs quota' still returns that the use is far from the quota 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 writes

150MB quota free
=> can still handle writes

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

Issue #1: Solution

- Dynamic qunit
  - 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
- Landed for 1.4.12 and 1.6.5

Issue #2: Quota overruns

- Client nodes cache dirty data
  - Up to max_dirty_mb (=32MB) per OSC
  - The client cannot get ENOSPC thanks to the grant cache
- 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

Issue #2: Solutions

- Workaround landed in 1.x.y
  - Ask the client to stop caching data sooner than later
  - Tunable via quota_sync_blk (bug16642)
  - Unfortunately, does not address all the cases
- Real solution
  - introducing some quota knowledge on the client
  - Grant quota space to client
  - Quota + grant could be given to clients as part of extent locks
  - Longer term solution requiring quite a lot of development

Issue #3: Adding OSTs

- Online OST addition is not handled properly
  - quotacheck/quotaon needs to be run on this new OST
  - but currently, this requires a full quotacheck :(
- Solution
  - Set up quota files at mkfs time
  - Trigger quotacheck on quotaon, if no quota file exists
  - Store on the MDS the state of OSTs

Issue #4: Enabling quotas by default

- Two ways to enable quotas automatically at start up
  - At mkfs time: --param quota_type=ug
  - With lctl conf_param
- The annoying thing is that this has to be set up for each target
- Solution
  - Add a global quota parameter
  - No real use case for using quota on a subset of OSTs
    - Unless we want to support quotas on specific OST pools

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

Quota support on DMU

- DMU now supports per uid/gid quota
  - Used to support quotas only on fileset
  - Quota accounting always enabled on DMU, so quotacheck is no longer needed
- Some changes to the lustre quota code needed
  - Need to interface with the DMU quota API
  - Accounting handled by DMU, but lustre is now in charge of handling quota limits and returning EDQUOT (currently handled by ldiskfs)

Thanks!!!

Johann Lombardi
johann@sun.com