Quota For Lustre

February 11, 2008

1 From Engineering Requirements Specification

1. Lustre can operate and enforce disk block quota and file quota.
2. Hard and soft quota are supported
3. Central management tools enable setting limits for users and initializing quota check operations
4. Quota are only needed for Linux 2.6

2 Specification of subsystems

Definition: An *operational quota file* is a quota database containing limits for some uid's and gid's which is being used to enforce quota. An *administrative quota file* is a similar database, but it is used for recovery and soft quota or administrative purposes.

2.1 Master & slaves

A node is a master for a uid or gid if the node holds the cluster wide limits (hard, soft, files, blocks & gracetimes) for that uid or gid in an administrative quota file. The administrative quota file is similar to normal ext3 quota file. The data structures and code for an administrative quota file API will be copied from the Linux VFS to ldiskfs and amended. Slave nodes (all other servers) only consider hard quota and only have operational quota files.

Note that a node may be a master for some uid's, gid's and a slave for others. Masters also have an operational quota file for enforcing hard quota. Master observe soft limits in the administrative file, based on grace times.

2.2 Acquire / release protocol

The master administrative quota file has two kinds of limits: total limits and limit acquired by all servers (administrative usage). Totoal limits are set by user,

administrative usage is initialized to zero and it's amended when master/slaves acquire or release quota.

Quota slaves can acquire from the master and release to the master qunits of disk space (>100MB typically, see ERS). Slaves do this to increase / lower their hard limits of operational file. Upon acquiring quota from a master the master's administrative usage are increased. Master can acqurie/release qunits, just like slaves, except that it is done locally.

On the master only, soft limits are enforced in obd layer based on the administrative quota file. Once administrative usage > administrative soft limit, the timer is activated.

2.3 Chown Operations

All objects associated with a file will have their owners set to that of the MDS inode. These chown operations occur in connection with file creation and chowning on the MDS and are asynchronous. There will also be enough space in the records to set an EA on the objects indicating the originating MDS, fileset and storage id of the inode. The arguments will contain the following - but the final format of the packet sent is subject to approval by management (it may be larger):

```
struct object_setattr_args {
      __u64 osa_mds_id;     /* to identify MDS */
      __u64 osa_fileset_id; /* part of the fid, tbd */
      __u64 osa_ino;    /* inode number on mds */
      __u64 osa_gen;    /* inode generation on mds */
      __u32 osa_uid;    /* owner of the file */
      __u32 osa_gid;    /* group of the file */
      __u64 osa_mds_transno;/* for recovery of mds rollback */
      __u64 osa_mds_last_committed;
      __u32 osa_mds_prev_uid; /* to undo things that didn't complete on the MDS */
      __u32 osa_mds_prev_gid;
}
```

2.4 Recovery

A recovery protocol for limits involves

Master recovery re-writing the administrative usage on the master node, based on the cluster-wide limits collected from slaves and master.

Slave recovery release unreasonable high quota limit.

Chown operations for objects will use llog recovery on the MDS (as it is used for unlinks).

MDS chown operations that are lost are not recovered at this point - but arguments to do so in the future are passed as above. The recovery from this is

fairly simple: the OST writes log operations for each chown operation containing the MDS transaction number and undo information. The MDS reports last committed transactions to the OST. During normal use these lead to cancellations of records leading up to that transaction. During recovery, all llog records following the record containing the transaction number will be used to undo the OST chown/chgrp operations.

For new files, removal of objects does already take place.

2.5 Configuration

A configuration protocol will initiate quota check operations, turn quota on, and set limits. All commands will be issued through lfs.

2.6 Disk fs handling

Disk file systems track quota usage. An interface between OSS and MDS and disk file systems will enable a check and adjustment of disk file system quota limits before operations proceed. Every node will try to acquire quota before proceeding. Every node will release quota after finishing. Acquire and release calls are tuned to anticipate use. Disk fs quota check handling will be possibly on busy file systems.

3 Use cases

Each use case is an interaction between a "user" and "system". For each use case we describe what subsystem forms the "user" and the "system". Use the logical components indicated in sections 3.1-3.4 below to describe the use cases. The purpose is to check that each of the use cases at a high level appears to execute successfully by using the components listed under 3.1-3.4. In some of the scenarios (e.g. 3.2 multiple use scenarios should be described, e.g. how is the slave-master protocol involved and how is the client - oss protocol involved).

3.1 Initialization operation

3.1.1 Changing owners

The following operations are done on a client, and it can be run on a live system:

Administrator get root priviliges on the file system

Administrator run 'lfs quotachog -i <mnt>'

1. <mnt> is a mount point of lustre filesystem
2. *quotachog* is a command of *lfs* which will do chown/chgrp, in case of concurrent operations by other processes, it can set *-i* option to ignore ENOENT error.

System *quotachog* will abort if change failed, and then report error, indicating what was searched etc. Generally user cannot ignore the error, and should fix it and redo the above, except that user can set *-i* option for *quotachog* to ignore ENOENT error.

3.1.2 Mounting existing file systems with quota support

Administrator file systems on all server nodes should be mounted with quota support, and this is enabled by default.

System all needed modules are loaded, and file systems are mounted with quota support.

Administrator run 'lfs quotacheck', it will initiate quota check on all MDS' and OSTs.

System on each node "quotacheck" will walk through the diskfs. When the check finishes, it will report the check status back to the initiator. If it fails, the error is listed.

Administrator user should fix the errors and recheck before preceeding to the next step.

Administrator run 'lfs quotaon', it will initiate quotaon on all MDS' and OSTs one by one.

System each node will start to check/handle quota. The status will be reported back to the initiator.

Administrator user should fix the errors if there are.

Administrator run 'lfs setquota', it will set limits in administrative quota file on the corresponding MDS master for the specified uid/gid.

System if previous limits(hardlimit & softlimit) for the uid/gid are zero, master will initialize quota on all slaves and local node, otherwise only modify the administrative quota file. Moreover, the limit info is saved in administrative quota file on master. The status will be reported to initiator.

Administrator if some nodes failed, generally user should not ignore the errors.

3.1.3 a new file system to a state where it is using quota

Like above, but only need three steps: 'lfs quotacheck', 'lfs quotaon' and 'lfs setquota'.

3.2 Normal use block quota

Demonstrate how quota are acquired and released during normal use through sequences of the API's and network calls defined in this document.

DESCRIBE CASES WHERE

1. A USER DOES THIS OR THAT: WHAT are the system responses
2. The client does this or that: what are the OSS & MDS responses
3. The OST does this or that, what are the obdfilter / diskfs reponses

3.2.1 Acquire quota

User	issues file write operation.
System	performs write successfully and returns the written bytes.
Client	makes IO requests to OSS.
OSS	acquires qunit if needed.
Master	increase usage in adminstrative file then reply to OSS with granted qunit.
OSS	updates local operational quota file, performs write operation and replies client the ~noquota flag.
OST	calls obd_commitrw to commit write.
Obdfilter	if not enough qunit, acquire qunit by dqacq rpc from master, updates local operational quota file after dqacq reply, then performs normal direct write.

3.2.2 Release quota

User	issues truncate or unlink operation.
System	performs the truncate/unlink operation and returns error code.
Client	makes OST_PUNCH or OST_DESTROY requests to OSS.
OSS	performs truncate/unlink on objects. release qunit to master if needed.
Master	decrease usage in administrative file and reply to OSS.
OSS	updates local operational quota file.

OST calls obd_destroy/obd_punch.

Obdfilter performs unlink/truncate on objects, if there is qunit to be released, release qunit by dqrel rpc to master then updates local operational quota file.

3.3 Running out of block quota

User issues file write operation.

System write fails and return EDQUOT. (but the pages in cache will be written successfully)

Client makes IO requests to OSS.

OSS acquires qunit from master.

Master reply noquota to OSS.

OSS fs write fails, rewrites pages from client cache forcibly, replies client the noquota flag and error code.

OST calls obd_commitrw to commit write.

Obdfilter acquiring qunit fails, then performs normal direct write and fails, and then rewrites the pages from client cache, returns error code and noquota flag to OST.

3.4 Freeing space to get under quota

The release steps are the same as those in 3.2.23.2.2.

User issues file write operation.

Client makes synchronous write rpc to OSS if there is noquota flag.

OSS performs fs write successfully, return client ~noquota flag.

Client clears noquota flag for this uid/gid.

3.5 Enforcing soft quota

3.5.1 Start soft quota timer

User issues file write/create operations.

System returns successfully.

Client makes file write/create requests to OSS/MDS.

OSS/MDS sends dqacq rpcs to get more quota from master.

Master starts the timer once administrative usage > administrative soft limit and grants qunit to OSS/MDS.

OSS/MDS write/create succeeds.

3.5.2 Soft quota timer goes off

User issues file write/create operations.

System returns EDQUOT.

Client makes file write/create requests to OSS/MDS.

OSS/MDS sends dqacq rpcs to get more quota from master.

Master returns noquota to OSS/MDS.

OSS/MDS write/create fails and returns error code to Client.

3.5.3 Stop soft quota timer

The release steps are the same as those in 3.2.23.2.2.

Slave calls dqrel rpc to release extra quota.

Master stops the timer once administrative usage <= administrative soft limit.

3.6 File quota on the MDS

For CMD, it is similiar to block quota described above. For b1_4, it is completely managed by MDS locally.

3.7 Listing quota

User runs 'lfs quota', it will make an rpc to the corresponding MDS master for the specified uid/gid.

System displays usage & limits related to quota for the uid/gid on all nodes in the cluster, if some nodes failed, reports the error to user.

3.8 Recovery of quota

just describe interaction initiator - response, no internals

3.8.1 Master recovery/Slave recovery

Master	enquires all slaves' operational limits by issuing a get limits rpc.
Slave	releases unreasonable high limits then replies with limit.
Master	updates usage of administrative quota file.

4 State considerations

4.1 Node state

4.2 Context state

5 Logic specification

The quota implementation falls into a few, almost separate, components.
ORDER OF IMPLEMENTATION

1. Administrative utilities, with sufficient flexibility to create unit test cases
2. Administrative quota file implementation
3. OSS enforcement of quota (can be tested separately)
4. client - OSS protocol
5. quota context
6. quota acquire release protocol
7. MDS-OST setattr calls
8. comprehensive testing of use cases
9. recovery protocol
10. soft limit

5.1 Administrative utilities

For all of the following commands it is probably useful to define a single data structure that has enough fields to hold all the data that needs to be transfered.

Top priority

1. All utilities are either:
   (a) file system ioctls - where non-standard Lustre specific info is needed (e.g. listing)

(b) standard quotactl interfaces

2. A lustre obd_iocontrol will allow an MDS to initiate quota check or quotaon operations on all OST's. It should be possible to issue this ioctl as a file system ioctl on a client, or giving an MDS device on an MDS. **NOTE:** This rpc can be the same as the master to slave recovery enquiry rpc defined below.

3. an obd_iocontrol and special lfs is needed to display usage & limits related to quota for a uid/gid on all nodes in the cluster. This needs to be added to lfs and need to be a command that can be issued from a file system client.

4. a command is needed to set the limits for a uid/gid, perhaps based on a template. The limits need to be set on the master and in the limit database. All slaves need to be notified that quota tracking for the uid/gid is now in effect (perhaps by increasing quota limits on the node to a non-zero value). Similarly it should be possible to disable quota for a uid / gid.

5. Documentation for all of these will be implemented as manual page extensions and as part of the Lustre Users Guide.

6. A chown.chgrp utility. Build a small c utility that stats a file and then issues the chown/chgrp system call to change the ownder/group on all files under the specified mount point. This is issued from a client. This can only be run after the MDS has been changed to incorporate part 3.3

5.2 Adminstrative quota file & disk file system quota

1. The administrative quota file will be a quota file similar to ext3 based quota files with the usual VFS determined tree format.

2. The VFS quota api will be adapted to enable the administrative commands to create quota files by name and operate on them without sb (super block) or dquot quota context arguments as required.

3. **(Design this, but implementation is second priority)** Quota check will be adapted to handle checking on a live file system, as follows:

   (a) if inodes are not checked in sequence order (1,2,3, etc) the following is probably not possible.

   (b) block all operations on an inode while it is being "checked".

   (c) account for quota on inodes that are already checked

   (d) do not account on inodes that are not yet checked

5.3 OSS enforcement

1. The direct I/O and truncate calls on the OSS will enforce quota

5.4 Client OST/MDT protocol

The following component can initially be implemented based on quota status codes returned by the disk file system. In due course the status of quota will be determined by the acquire calls made in the OST or obdfilter.

1. All writes functions executed on OST's track quota for newly allocated space.
2. If a client flushes a page cache to an OST the data will be written (even if quota are exceeded). For this root privilege is needed - since only v2 quota format is supported, root always has the right of exceeding quota limits.
3. If a client exceeds quota, a return code will indicate that the for that further writes for files owned by that uid/gid must now be done synchronously.
4. If quota limits on the OSS are sufficient again, through removal of files or enlarging limits, the flag must be cleared.
5. For MDC file quota are currently handled synchronously on the server.

5.5 Quota context and server quota enforcement

1. The MDS will automatically track block quota associated with directories. It is important the llog files are owned by root users and not subject to quota
2. For root owned files, Lustre quota should not be enabled (there are too many administratively controlled root-owned files right now).
3. There will be an active **quota context** for a uid or gid for which quota operations are in progress. Processes acquiring quota will find the context for that user or group and wait on the context intelligently and not all fire RPC's to the master. The context should also intelligently handle recovery operations running concurrently with normal quota use.

5.6 Slave to Master acquire / release protocol

1. Tunables
   (a) All servers will have tunables for qunits and early acquisition of more qunits.

   (b) The tunables can be set to configurable values through lconf, one set of values for slave behavior, one for master behavior each separated for OSS nodes, one for MDS nodes, as part of the configuration zeroconfig llog.

   (c) The tunables can also be adjusted dynamically in /proc.

   (d) Adjusting through proc only is not acceptable.

2. There will be a function that determines the master node for a given uid or gid. For the 1.4 branch this function is always returning the MDS, but it will be designed to make it easy to adapt to clustered metadata.

3. There will be dqacq and dqrel rpc's initiated by slave nodes. The code will be organized so that it can be run on slave OSS and slave MDS nodes without modification. These functions will increase / decrease the local limits and administrative usage on master.

4. A unit test program will run a collection of not less than 3 slaves and a master through a sequence of interesting acquisitions and releases.

5.7 Full integration and system testing

1. Full unit tests for all components.

2. Demonstrate successful handling of recovery from exceeding soft and hard limits.

5.8 MDS - OST setattr calls

1. When the MDS creates or chown a file it will queue an asynchronous obd_setattr rpc to the OST that:

   (a) changes the owner/group of the objects for the file.

   (b) transfers the storage id (ask Yury for data type) to the OSS (this is in the create case only). It writes the storage id in an EA.

2. The obd_setattr calls will be journaled almost exactly like mds_unlink calls in an llog (except that for unlink presently the client unlinks the objects) and records will be canceled when the setattr commands commit to disk on the OST.

3. The obd_setattr rpc's will be queued on an RPC set for asynchronous completion, i.e. the MDS will reply to the client without waiting for the result. The simple strategy ("chown, even if user goes over quota", see ERS) will be followed.

4. For this part not more than 4 (four) lines of code may be added to mds_open. Adding 0 lines to this function (the longest in Lustre) would be better.

5. Demonstrate handling recovery of 300,000 orphaned chown operations while the cluster is in use already.

5.9 Server Node Recovery

Note: in CMD nodes will be slaves for some uids and masters for others. The algorithm outlined here handles the general case.

1. Nodes will recovery quota asynchronously, ie. they will start normal operations, without waiting for quota recovery to complete.
2. **Slave recovery initiation:**
   (a) Slave recovery is initiated on a per-connection basis
      i. Upon obtaining a new connection to a server node that can be a master during normal operations
      ii. Upon entering normal operations while connections are present
   (b) The recovery is aborted if a connection fails.
   (c) A collection of threads is needed to handle this recovery
   (d) The quota file handling should be sufficiently concurrent that multiple connections can recover in parallel
3. **Slave recovery:**
   (a) During normal use the node will iterate through all the users and groups in the operational quota file.
   (b) If the connection is not one to the master for this uid/gid go to the next uid/gid.
   (c) If such a uid/gid is also found in the node's administrative quota file, this node is the master for that id and this id will be skipped, else continue.
   (d) Release unreasonably high limits for this uid/gid.
   (e) The contexts used for updating quota from the filter should be design so that these releases can be made concurrent with normal use.
4. **Master recovery initiation**
   (a) Master recovery requires connections to all other servers, it is initiated:
      i. If upon entering normal operations all connections are present
      ii. If during normal operation all connections reach a usable state
   (b) It is aborted if any connection fails during master recovery
5. **Master recovery:**

(a) During normal use the master will iterate through the administrative quota file.

(b) It will lock quota operations on the master for that uid.

(c) For each uid/gid found it will make a new quota related master to slave RPC to all other servers and ask for the current limit (and usage).

(d) If a response is obtained from all nodes, the operational limit on the master node is updated so that the sum of all operational limits is the clusterwide administrative limit.

(e) '

(f) If a response is not obtained from all servers, abort.

5.10 Soft Limits

Soft quota is not enforced in fs layer on master or slave. It's only enforced in obd layer on Master:

1. The grace time and soft start time will be kept in adminstrative file.
2. Master monitor the administrative usage on each qunit acquire/release handling: log the soft start time once the administrative usage >= administrative soft limit, clear the soft start time once the administrative usage < administrative soft limit.
3. Master will reject any qunit acquire request if soft start time + grace time < current time.

Make sure we have unit tests and integration and system tests that verify this comprehensively.

6 Changelog

2005/01/29 First draft. Based on review of Zhaohongs writings and ERS.

2005/02/06 Second draft, much more detail to aid the team