OST Pool Based Quota

Version	Date	Comment	Author
1.0	11/26/2013	Initial revision	lixi@ddn.com

1 Introduction

The rapid surge in the overall number of OST within a Lustre clusters increasingly raises questions of Lustre usability and management. OST pools enable administrators to group OSTs together for both more flexible, and more controlled, striping which might help to reduce the difficulty in very large Lustre file system. But, at present, quota support for OST pools is not available, which limits the usefulness of the OST pool feature. Fortunately, the existing quota framework of Lustre is powerful and flexible; we have extended this framework to enable pool-based quota settings in Lustre; this document describes the design and implementation of OST pool based quota in Lustre.

2 Motivation

Quota is very basic, yet extremely useful storage management mechanism for file systems. Quota is typically implemented on a per-user or per-group basis. However, in a massive distributed file system, per-user or per-group quota is no more sufficient to account for the use scenarios encountered by storage administrators. Imagine multiple users or groups collaborating on a shared project, and saving all their files to the same directory. Administrator might want to enforce a total limit on the disk usage for the entire project, or directory. The traditional user- or group-based is insufficient to account for this scenario.

In order to address this kind of problem, a number file systems support quotas for separate sub-sets of the file system. For example, XFS supports per-directory or per-project quota by managing the disk usage of directory hierarchies associated with a specific project. GPFS supports fileset-based quota, which limit the scope of quota enforcement an individual fileset boundaries. A patch for sub-tree quota support in ext4 has been available for years.

Currently, the Lustre file system does not provide mechanism to enforce quota limits on projects or directories and the assumption has been that directory-based quotas would require support from lower-level file systems in Lustre. We have argued that the notion of an “OST pool” enables the implementation of more fine-grained quota, i.e. OST-pool based quota. Pool-based quotas are a much more straightforward to implement than directory based quota but, also, a pool-based quota framework can be used to define quotas on a given directory associated with an OST-pool. This can

be seen as an enhancement of the existing user/group quota framework: Administrator can set quota limits for user/group to specific OST pools, thus signify broadening the possibilities for quota settings within Lustre.

3 Requirements

The design and implementation of OST pool-based quota should satisfy following list of requirements:

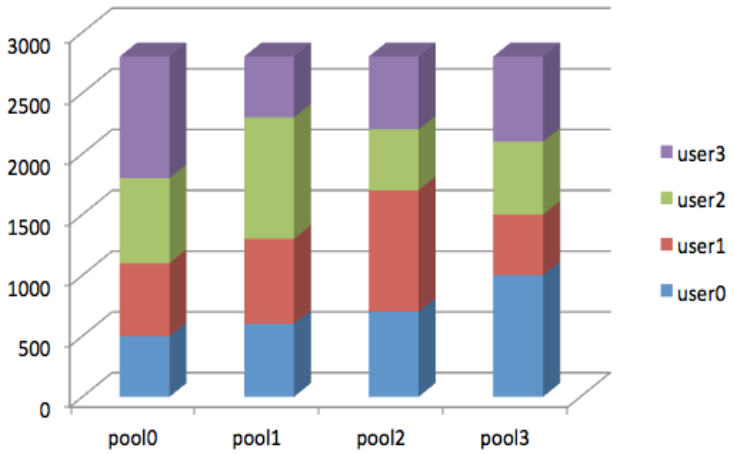
- Integration in the current quota framework. OST pool-based quota should be able to enforce both block and inode quotas, but also support hard and soft limits. The current user/group accounting should also be supported.
- Full support for pools. OST pool-based quota should support separate quotas for users/groups for each pool. Dynamic runtime changes of pools should be supported. OST pool-based quota should handle this without issues.
- No significant performance impact.

4 Design and implementation

The notion of an "OST pool" allows administrators to associate OSTs into different groups or "pools". With quota support for OST-pools, we can set different quotas for individual OST pool. The quota support for OST pools should follow the generic rules for OST pools:

- An OST can be a member of multiple pools.
- No ordering of OSTs in a pool is defined or implied.
- Stripe allocation within a pool follows the same rules as the normal stripe allocation.
- OST membership in a pool is flexible and can change over time.

In our implementation, the quotas in different OST pools do not impact each other. There is no inclusion, grouping, or any other kind of relationship between the quotas set on different OST pools. From the perspective of quotas, the entire file system is divided into separate portions, one portion for each OST pool. Though OST pools may share the same OSTs (according to the rules of OST pool feature), their space usage accounting and limit enforcement are totally independent. Below is a simple example:



Graph: User/Group quotas setting on different OST pools are independent.

In the current implementation of the Lustre quota system, quotas are defined for the entire file system. The OST pool-based quota feature enables administrators to set individual quotas for each OST pool. Note, however, that the support of hierarchical quotas, i.e. both quotas for the entire file system and for OST pools, is beyond the scope of this feature. It is possible to enforce both quotas for the entire file system and for the OST pools at the same time, but it is neither efficient nor straightforward. In our implementation, the space usage and limit of a given user/group in the entire file system is calculated by summing up the limits and usage of all individual OST pools.

For simplicity and compatibility with old versions of Lustre, we define a "default OST pool". The default OST pool is different from other OST pools. Its ID is zero. It includes all of the OSTs in the system and does not have an explicit name (as other OST pools do), and this is why we refer it as the default pool. The default pool exists all the time and can never be removed from system. The concept of a default OST pool simplifies the compatibility problems of the proposed OST pool-based quota feature significantly.

Quota accounting and enforcement is always turned on. As soon as a pool is created, its space accounting begins, thus former APIs which were used to turn file system quotas on or off are inactivated.

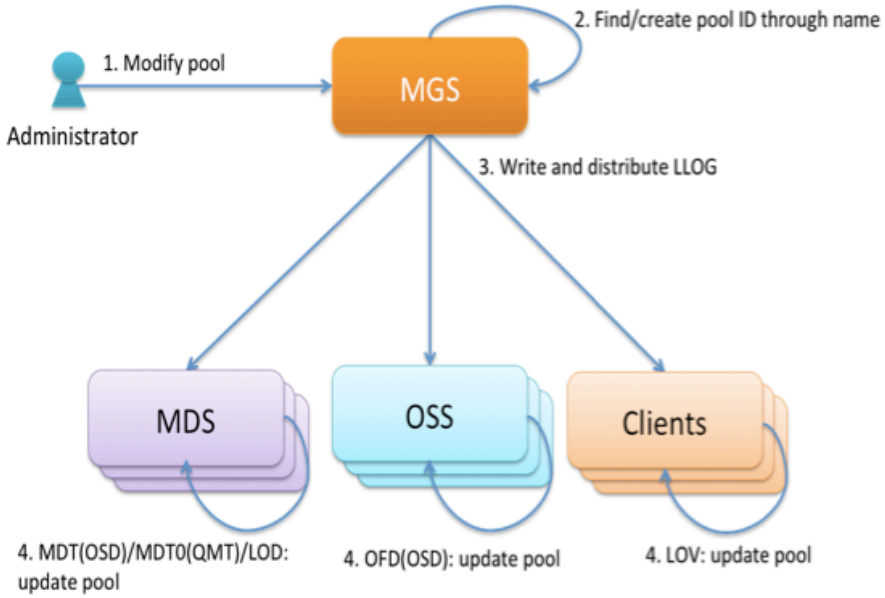
4.1 LLOG processing

The configurations of all OST pools are saved in MDS LLOG. Taking advantage of the LLOG mechanism, the necessary information about OST pool configuration is distributed to the MGCs.

One of important information that the MGCs need to propagate to the OST pools are the pool IDs. In our implementation, OST pools can be identified not only by their names, but also by a unique 16-bit pool ID. The ID is assigned when the OST pool is allocated by calculating a hash value of the pool name. It is guaranteed that the ID is system-wide unique so that the servers can verify which pool a given operation belongs to. The pool ID is essential because it is not efficient to transfer pool names

through the network all the time.

The following graphic describes the processing of LLOG records of OST pools. When an administrator modifies the OST pool configuration, the MGS will find/create the pool ID of the pool based on the type of operation. If the record is LCFG_POOL_NEW, then a pool ID will be allocated for the newly created OST pool. Each pool configuration record contains both the pool name and its ID. The process of OST pool record happens not only when the administrator changes the pool configuration but also after the MGS service starts, because LLOG on the MGS becomes active.



Graph: Basic flow for pool-based quota.

As a part of the LLOG process, the MGCs will update their views about the OST pool configurations. A centralized server holding the cluster-wide limits, the QMT service runs on the first MDT. When a new OST pool is created, the corresponding quota files is created on the QMT so that QMT is ready for the global quota enforcement of the pool.

As quota slaves, all the QSDs on OSTs and MDTs need to update their configuration too whenever there is change of OST pool. A QSD instance is allocated and started for each pool when the pool is created.

4.2 Quota acquisition/release

When a quota slave queries quota or acquires/releases quota space from the QMT, the pool ID is packed into the global index FID. The QMT extracts the pool ID from the FID and thus know which OST pool the request is operating on.

4.3 OST/MDT objects

In order to enforce quotas for each OST pool, it is essential for a quota slave to know which pool an object belongs to. A new extended attribute is used to save the pool ID, i.e. XATTR_NAME_POOL. For simplicity of code, this extended attribute exist not only for OST objects but also for MDT objects, though the pool name can be extracted from the extended attribute named XATTR_NAME_LOV. After a server reboots, the pool ID is extracted from the extended attribute when the object is accessed and cached in memory. With a given pool ID, the corresponding quota instance of QSD can be found in order to account for the disk space consumed by the object and to enforce quota limits.

When an object on a MDT is created, its pool ID is obtained by searching the pool name, whereas the pool ID and extended attribute of an object on an OST is only set when a client updates the UID/GID and pool ID of the object for the first time. The OST extracts the pool ID from the request and saves it as extended attribute of the accessed object. In any case, the pool ID is initialized before the objects consume any disk space, so that space usage is accounted correctly for each pool.

5 Compatibility

The pool based quota patch introduces a number of changes to the Lustre quota framework, including the following:

- LLOG record format of pool configuration. The pool based quota patch adds the pool ID field in every record of a pool configuration.
- New LLOG records of pools. All OSDs on MDTs and OSTs need to be notified of the pool configuration change. The patch also adds new records for the QMT.
- Disk format of quota files. The on-disk directory structure for the quota slave has been changed in order to support pool based quota.
- Extended attribute of objects. A new extended attribute is used to save the pool ID of an object.
- Wire format. In order to transfer pool ID and other information through RPCs, certain on-ware formats needed to be changed.
- Proc virtual file system interface. The proc entry for both QSD instance and QMT instance of pools were changed.
- Utility interface for quota control. The lfs and lctl commands are changed so as to support pool-based quota.

These changes introduce compatibility problems with old versions of Lustre. Currently, the code basis to maintain compatibility is not finished, and we believe some additional discussion is needed.

First of all, when updating to the new code base with pool based quota support, all servers need to be updated since there are too many on wire and on disk changes

that could cause unexpected problems.

If all Lustre clients and servers have updated to the new version, compatibility problems are caused by three kinds of on-disk change, i.e. LLOG changes, quota file changes, and new extended attributes of objects.

While a transparent upgrade process to fully support pool based quota is desirable, significant efforts would be necessary and it is not clear at this time whether an upgrade is always possible. a simpler way is, thus, to rely on external (i.e. manual) help in the upgrade progress.

In order to avoid LLOG problems, before the upgrade starts, the administrator should delete all pool definitions so as to leave no pool records in LLOG. It is simple and necessary for MGS codes to ignore the incomplete pool records without any pool ID field. In another word, if the administrator forgets this step, all pool definitions will be discarded in the new version as well. Since former pool definition in the old system is considered as obsolete, a version ID might be added to the new pool field in the XATTR_NAME_LOV extended attribute so as to prevent the confusing output of the 'lfs getstripe' command on an old file.

As mentioned before, a default OST pool is always present in the new system. All quota limits in an old Lustre file system will be inherited transparently as the limits of the default OST pool. However, there is a semantic change between the old system and new system quota i.e. the former limits are for the entire system and the new limits are enforced on the default OST pool only. This implies that all of the disk space consumed by existing files is considered to be equivalent to the disk usage of the default OST pool too. This is straightforward and should not cause any problems because the format of the quota file on quota master is not changed.

All of the existing files on the old file system are considered to belong to the default OST pool thus, if an old object is found to miss the XATTR_NAME_POOL extended attribute, the value of the extended attribute will be set automatically to zero.

It is not rare for a “old” client to connect to a newly updated file system. However, this might cause problem. When a client creates a new file on a non-default OST pool and writes some data into it, the correct pool ID should be sent along with the correct UID/GID. However, the old client won’t even try to obtain the correct pool ID.

The most obvious solution to this problem is to return a failure to the client whenever the correct pool ID is not given. But this solution is not friendly to the old client and might confuse users because, since only writing to newly created files will hit this problem. Another solution is to set a specific pool ID to the object when no valid pool ID is given. This ID is considered to be the same with the default pool ID, i.e. zero. But whenever it is possible, e.g. a new client accesses the file carrying the correct pool ID, the pool ID of the object will be updated to the correct one. At the same time, the space usage will be changed to the correct pool, just like the processes of updating the UID/GIDs of an object. While not a perfect solution either, it will basically cover the problem.

6 Usage

Lustre utilities have been adopted to support pool-based quota. As a first step, create and manage OST pools using normal utilities for pool management:

lctl pool_new fsname.pool1

pool_add server1.pool_1 OST0000

'lfs setquota' command has been changed so that we can assign a pool name with '-p <pool_name>' argument while setting quotas to pools. If this argument is not given, the pool is considered to be a default pool.

lfs setquota … [-p <pool-name>] <filesystem>

lfs setquota --block-hardlimit 2097152 -u user1 -p pool_1 /mnt/lustre

lfs setquota --block-hardlimit 1048576 -u user1 /mnt/lustre

'lfs setquota' command has a similar argument. With '-p <pool_name>', we can display quotas and disk usages of OST pools

lfs quota … [-p <pool-name>] <filesystem>

lfs quota -u user1 -p pool_1 /mnt/lustre/

lfs quota -u user1 /mnt/lustre/

'lfs setstripe' command is used to associate directories/files with OST pools in the usual fashion. Then, the quota limits are enforced on these directories/files:

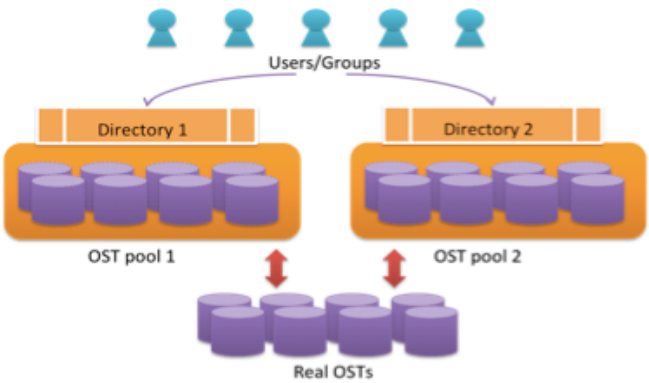
lfs setstripe <filename|dirname> --pool|-p pool-name

lfs setstripe -p pool_1 /mnt/lustre/dir1

7 Use Cases

7.1 Users/group quotas for directories

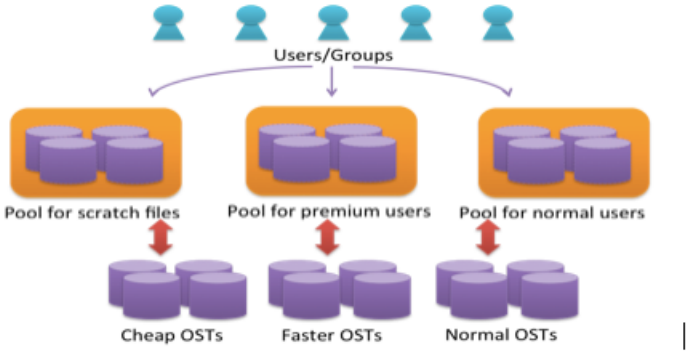
We believe that there exist a number of interesting use cases for pool-based quotas. One immediate use case is quotas to different directories. Below is a graphic that shows the basic idea:



Two different OST pools on the same physical OSTs can take advantage of the pool-based quota support and different quota limits can be set to the pools. By defining directory stripes corresponding to OST pools, it is possible to define quota limits for directories that share the same physical OSTs.

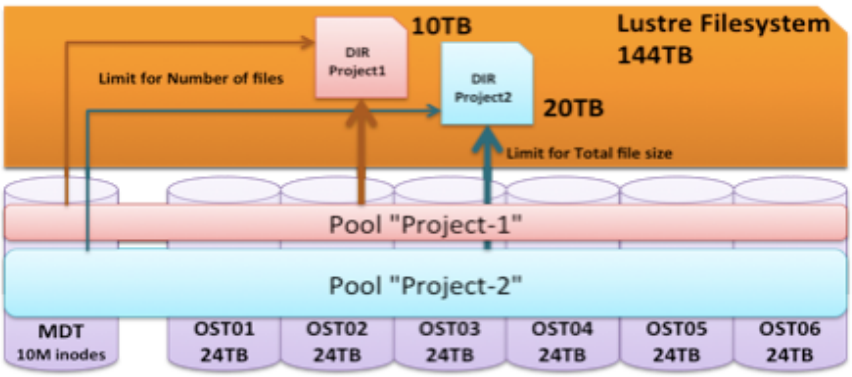
7.2 Quota for different kinds of OSTs

Another use case is to classify OSTs into different pools according to certain characteristics, e.g. such as the type of storage media. The following shows an example. OSTs are classified into three pools, according to access speed; one pool with cheap OSTs for scratch files, another one with faster OSTs for premium users, and one pool with normal OSTs for standard users. With the help of pool-based quota, the usage limits for different pools can be set differently for specific users. For example, quotas can be set in a way that users with high priority can use more capacity on high-speed storage spaces than users with lower priority.



7.3 Directory or project based quota

As mentioned before, directory or project based quota are very useful for collaboration or cloud storage. What follows is an example of using pool-based quota to limit the space usage of different projects. In this case, it is actually necessary need to account the space usages of pools in total along with users/group-based accounting.



8 Future work

We have built the basic framework for pool-based quota, but there is some work to do in order to finish the project, notably the following:

- Compatibility with older versions. Extra codes should be added to handle the compatibility problem with older versions.
- Test cases. A number of test cases for pool-based quota should be added to verify correctness and efficiency.
- Space accounting for pools along with users/groups. This will enable directory or project based quota, and will significantly extend use cases for pool-based quotas.
- Clustered meta-data support. If the MDT pool is useable in DNE, MDT-pool based quota would be straightforward, and would provide for a powerful tool to manage the usage of metadata.

9 Acknowledgments

We would like to express our gratitude to all the community developers who have contributed to this project through their advice, and we would especially like to thank those who have reviewed our patches!