Lustre Enhancements – Technical White Paper

Implement hash tables to scale export lookups

Author	Date	Description of Document Change	Client Approval By	Client Approval Date
green	May 07, 2008	Create the document		
Nirant	May 09, 2008	Review and updates		
green	June 05, 2008	More implementation details		

Problem Statement

When multiple clients connect to a service, the resulting exports are linked into several lists. Lookup traversal of such linked lists (for purposes of finding the exports on subsequent reconnects, disconnects and admin-induced evictions) consumes a lot of cpu time, especially when many clients are connected. Similar problem exists for connection lookups.

Approach

Based on the above analysis, the exports lookup by UUID or NID needed a more scalable implementation. A decision was made to create a generic hash table implementation and use that implementation for export and connection lookup purposes.

Hash table is a method of finding data where every element stored in the table gets a "key", which is calculated based on element properties. One big linked list is then split into a configurable amount of smaller lists, one list per every key possible. Each new element is now being inserted into a much smaller list now, and for lookups given the search criteria, a key is calculated and much smaller list needs to be iterated to find the matching elements. This results in vastly improved lookup speeds.

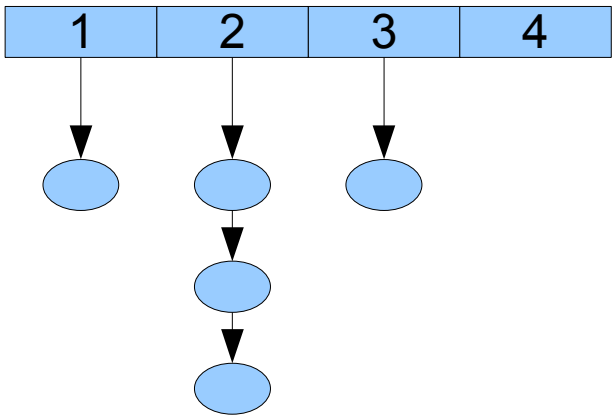

Figure 1: Hash table

Above figure demonstrates hash table concept. When a search is done for a specific element, we only search a list in a specific bucket, corresponding to a key calculated from element properties. The search time could be reduced by a factor of number of buckets in a hash table, subject to good hash functions (key calculation) with low amount of collisions for the chosen data.

Actual Implementation

Implementation of hash tables now lives in obdclass/class_hash.c, and actual definitions of structures and export methods live in include/class_hash.h.

Actual hashtable is defined by lustre_class_hash_body structure, that has these members:

- hashname – name of the hashtable
- lchb_hash_max_size – number of hash buckets in the table
- lchb_hash_operations – list of operations with hashtable and objects there
- lchb_lock – lock to protect access to the table

Every bucket just contains linked list of objects and lock to protect list manipulations.

List of operations on hastables and objects:

- lustre_hashfn – calculates hash value for a given object
- lustre_hash_key_compare – compares if provided key with key of provided object
- lustre_hash_object_refcount_get – obtains one reference count on an object
- lustre_hash_object_refcount_put – releases one reference count on an object

Operations hash table users can perform on the table:

- lustre_hash_init – create and initialize new hashtable
- lustre_hash_additem_unique – add an item with unique key into table (or refuse to add, if item with such a key already exists in the table)
- lustre_hash_findadd_unique – add an item with unique key or return existing item in the table with same key, if already present there.
- lustre_hash_additem – adds an item into the table
- lustre_hash_delitem_by_key – deletes one item with matching key from the table
- lustre_hash_delitem – deletes item from the table.
- lustre_hash_bucket_iterate – iterates through all objects in a bucket matching provided key, calling provided callback
- lustre_hash_iterate_all – iterates through all objects in the table, calling provided callback.

- lustre_hash_get_object_by_key – returns a pointer to referenced object from a table, matched by key.

Actual hash used for uuid, nid and connection hashes is djb2 hash algorithm, that is believed to be pretty good for these kinds of values. Actual hash-buckets used for these hashes is 128.

Test cases and Results

A test was performed by Cray to start up a large scale job (19180 client nodes.)

Before the hash table patch, such a job made every OST CPU-bound taking over 180 seconds to just connect to all the nodes. After the patch was applied, connection times dropped significantly to around 2 seconds.

Before Patch	After Patch
180s	2s

Table 1: Before and after Results

Conclusions & Future work

UUID Hash code implementation is functional and is included in all Lustre releases, as it provides significant performance benefits when running jobs at large scale. Since the time of inclusion of this code, other parts of Lustre have started to use it as well, such as the Quota code.