Subtree Locks

Alexander Zarochentsev , Vladimir Saveliev

(Started: 2008.02.01)

Contents

1 Introduction

Subtree lock is a lock on a directory which protects an entire namespace (or its part) rooted at that directory. Subtree lock is supposed to be optimal for workloads where clients work in isolated directories and to not make things worse in highly contended workloads by resorting to current client-server locking protocol.

2 Requirements

Performance reduce lock RPC traffic for STL-locked objects.

Scalability reduce load of DLM server and memory consumption on servers and clients

Correctness provide a correct interaction between STLs and ordinary DLM locks.

Usability usability to other components (WBC, disconnected operations).

3 Functional specification

3.1 General

When a directory namespace is accessed by one client mostly (user works in home directory, for example) it may be useful to grant the client with a single lock for whole subtree.

Advantages:

- reduced DLM load

- reduced server and client memory consumption
- fewer lock RPCs

Disadvantages:

- a MD server does not remember which objects are cached by STL holder, so it has to send BAST to STL holder whenever it acts on behave of other clients
- other clients are to care about being inside of outsider's STLs

3.1.1 Note

File data are protected with IBIT lock, which is independent of subtree lock but is discribed in this document.

3.2 LDLM

Subtree lock is a new type of lock.

3.3 Metadata protection

STL protects all metadata objects in the namespace below STL root except for

- objects explicitly locked by ordinary locks
- mount points
- metadata of hard linked files

When STL holder caches an object under its STL from MD server:

- if the object is not protected, ordinary lock mode is applied
- if the object is protected, MD server just sends the object to STL holder

3.4 File data protection

- When, in the absence of concurrent users, a file is opened on an MDS, its first user is granted special IBIT lock, permitting an exclusive access to all file sub-objects on OSTs.
- When conflicting access occurs, this lock is revoked, and users switch to the usual extent based locking.

3.4.1 Note

IBIT lock applies to files which have more than one link as well.

3.5 Nested subtree locks

Subtree locks can be nested. A client can not acquire a subtree lock on an object if that object is under subtree lock owned by another client. Ordinary lock is acquired in this case. For example, if client 1 holds subtree lock on /a/b/c, then client 2 can acquire subtree lock on /a, but can not acquire subtree lock on /a/b/c/d/e. This results in that objects cached under inner subtree lock are never cached under outer one. In the above example, client 2 can cache under its subtree lock all objects in /a but /a/b/c and objects below it.

3.5.1 Note

An object can be cached only under the nearest subtree lock if there are several subtree locks above.

3.6 Policy

3.6.1 Client policy

On lookup a client always requests a subtree lock for an object being lookuped.

There are three exceptions. A subtree lock is not requested when

- the parent directory is already protected by subtree lock held by the client
- the parent directory is protected by subtree lock of another client.
- it might make no sense to acquire subtree locks on objects involved into rename.

More details are in Use cases section (4.6.1) and in Logic Specification section (5.4.1).

3.6.2 Server policy

If a client requests for subtree lock on an object, MD server grants it if there were no recent accesses to the object from other clients.

More details are in Use cases section (4.6.2) and in Logic Specification section (5.4.2).

3.6.3 Reaction on a BAST

STL holder may react differently on a BAST. Possible actions are:

- If an object in conflict is not cached by STL holder - nothing is to be done
- if the object in conflict is not modified, drop it from the STL cache
- STL cancel
- STL split into STLs of subdirectories
- if the object in conflict is modified, flush it together with dependent updates and drop the object from the STL cache

MD server may explicitly request STL holder to perform certain action, lock cancel, for example. A BAST may contain auxiliary information, FID of object in conflict, for example.

3.7 Clustered Meta Data (CMD)

Subtree locks should apply transparently to CMD case. Related issues are:

1. FID of an object allows to determine MD server where the object is stored.
2. When client lookups for an object in a directory, it includes into the request an information about existence of any subtree locks above the parent directory.
3. A subtree lock is created on MD server where inode of subtree lock root is stored.
4. Instances of a subtree lock on other MD servers storing elementes of directory protected by the subtree lock are not needed.

3.8 Lock revalidation

3.8.1 Summary

When a client acquires a lock on an object without downward lookup (the most comon case is accessing “.” and “..” via FIDs), the client has to make sure that the object has not been recently cached under STL by other client . This problem is solved by lock revalidation mechanism.

3.8.2 Scenario where acquired lock is to be revalidated

- A client C1 holds ordinary lock on an object O1 (it did chdir(/a/b/c/d/e), O1 is inode of /a/b/c/d/e) and is idle for some time.
- Another client C2 does something with O1 (ls -l /a/b/c/d/e), MD server sends a BAST to C1 and C1 cancels the lock of O1.
- C2 is not interested anymore in O1, so it drops the lock.
- Yet another client C3 acquires subtree lock on /a/b and caches and possibly changes (if under WBC) objects under /a/b including /a/b/c/d/e (the object O1).
- C1 continues with stat("."). It sees that the lock on O1 is canceled, so it goes to MD server with FID of O1 and acquires the lock on O1.

As long as C3 has changed O1 and MDS is not aware of that, the MDS granted C1 with lock on O1 incorrectly. The lock on O1 had to be revalidated. The procedure of revalidation is explained briefly in 3.8.3 and the pseudo code is in 5.6.

3.8.3 Revalidation procedure outline

A client gets a lock (L1) on an object (O1) and has to revaldiate it. The revalidation procedure goes up getting FIDs of ".."s from corresponding directory entries taking ordinary locks on MDS on inodes of parent directories. Even though that way up is not necesasry uptodate, it will end up with a STL root or with the filesystem root.

- if the way up reached the filesystem root directory - check the lock L1. If it is not revoked, it is valid. Otherwise, the revalidation repeats.
- if STL is found - make sure that STL holder does not cache the object O1. For simplicity, the STL can be revoked completely. If STL holder does not cache O1, there is no need for revocation.
  - Lock L1 has some specificity: it is not revoked when subtree lock holder flushes its changes to O1, but it is revoked when some client caches the object.
- Reread the object and check the lock L1. If L1 is not revoked, it is valid. Otherwise, the revalidation repeats.

4 Use cases

4.1 Acquire subtree lock

A client lookups for name in a directory **D**. The client does not hold subtree lock on any current working directory component. Server may grant either subtree lock or ordinary lock.

Client:

- determine which MDS to send lookup request to based on FID of **D**
- send lookup request to the MDS
  data included into the request:
  - FID of **D**
  - name to look for
  - flag indicating that subtree lock is desired

  The code path for that: ll_lookup_it -> md_intent_lock -> mdc_enqueue.

MD server:

- get lookup lock on **D**
- lookup for name in **D**, if name is not found - return ENOENT
- obtain FID of object being lookuped
- decide which lock (subtree lock or ordinary lock) the client can be granted with
- get either subtree lock or ordinary lock on the object for the client
- put lookup lock on **D**
- return to the client
  - attributes of the found object
  - lock to the object

Client:

- create the lock instance on the client and link inode of found object to that lock

4.2 Object access under subtree lock

A client holds subtree lock on a directory **D** and lookups for name in a directory **S**. Both the parent directory **S** and an object being lookuped are both protected by the subtree lock.

Client:

- determine which MDS to send lookup request to based on FID of **S**
- send lookup request to the MDS

  data included into the request:

  - FID of **S**
  - name to look for
  - flag that the lookup is performed under subtree lock

MD server:

- look for name in **S**, if name is not found - return ENOENT
- obtain FID of the object being lookuped
- get object attributes
- return to the client
  - attributes of the found object

Client:

- link inode of found object to the subtree lock under which the lookup was performed

4.2.1 Note

Should the subtree lock refcount be incremented on the client before sending request to MDS? If yes, that would allow to not care about race with subtree lock cancelation.

4.3 Concurrent lookup

This is the same as 4.7 Callback to ordinary lock.

4.4 Access to ".."

A client goes up in current working directory using dentry cache.

- get inode of parent directory using d_parent of dentry which represents current working directory
- if lock on the inode is not canceled yet - there is no problem
- if lock is canceled - lock revalidation is needed. See 3.8 for details.

4.5 CMD

4.5.1 Lookup in subtree lock

A client holds subtree lock on a directory **D** and lookups in a subdirectory **S** of the subtree locked directory. **S** is not locked.

- increment refcount on the subtree lock
- determine, which MDS to send lookup request to using FID of **S**. That MDS does not have to be the MDS where subtree lock is set.
- include to request a notion that the lookup goes under the subtree lock and send the request
- as long as **S** is not locked by other clients, it is protected by subtree lock. MDS looks for the name and returns FID of object being lookuped
- attributes of object are obtained without additional locking (if it is not locked by other clients)
- on lookup completion decrement refcount of the subtree lock

4.5.2 Lookup under outsider's subtree lock

A client lookups in a directory which is under outsider's subtree lock. The client knows which client is owner of the subtree lock.

- determine, which MDS to send lookup request to using FID of the directory
- include to request a notion that the lookup goes under the outsider's subtree lock and information about subtree lock holder
- MDS sends a BAST to the subtree lock holder. It is possible to specify the object of interest
- subtree lock holder reacts on the BAST. It may take appropriate action listed in 3.6.3
- MDS grants ordinary lock to the object

4.6 Policy

4.6.1 Client policy

A client is about to lookup for an object in a directory. The client has to decide which lock to request from a MD server.

The decision is made based on locking state of inode of the directory. The inode of the parent directory can be in one of the following locking states:

1. protected by subtree lock held by the client

   The client requests for access to lookuped object under subtree lock

2. protected by ordinary lock because MD server granted ordinary lock

   The client requests subtree lock for lookuped object

3. protected by ordinary lock and is marked as “under outsider’s subtree lock”

   The client requests ordinary lock for lookuped object

   On success the client has to mark the object as “under outsider’s subtree lock”

4.6.2 Server policy

A server receives a request to grant a subtree lock on an object. The server has to decide whether subtree lock should be granted.

- there are objects to which subtree locks do not apply
- subtree lock granting decision is made based on statistics of accesses to the object

More details are in 5.4.2.

4.7 Callback to ordinary lock

A client holds subtree lock on a directory and lookups downward in the protected directory, either parent directory or object being lookuped are protected with ordinary lock.

for example, subtree lock is on /a/b/c, lookup in directory /a/b/c/d for “e”, “d” or “e” are locked by ordinary lock

Client:

1. Based on FID of parent directory the client determines which MDS to send lookup request to.

2. The client sends lookup request to the MDS.
   Data included into the request:
   - FID of parent directory: FID of /a/b/c/d
   - name to look for: “e”
   - flag that the lookup is performed under subtree lock

MD server:

- if parent directory is locked by other client
  - acquire ordinary lock on the parent directory
  - set a flag saying that lookup has to continue with ordinary locking
- look for name in the parent directory, if name is not found - return ENOENT
- obtain FID of the object being lookuped
- if ordinary lock is to be used or the object is locked by other client
  - acquire ordinary lock on the object
- if ordinary lock was acquired on the parent directory lock
- MDS returns to the client
  - if ordinary lock was acquired on the parent directory
    - attributes of the parent directry
    - lock to the parent directory
  - attributes of the found object
  - lock to the object if it was acquired
  - NB: if ordinary lock was used in the lookup - force the client to continue lookups with ordinary locks

Client:

- adjust client policy to use ordinary locking for lookups below.

4.8 Callback to subtree lock

A client lookups downward, there are subtree locks below. While the client requests subtree locks, MDS grants ordinary locks only.

for example, the client lookups for ./**D**, there is other client which holds subtree lock on **D**

Client:

1. Based on FID of parent directory ("**.**") the client determines which MDS to send lookup request to.
2. The client sends lookup request to the MDS.

   Data included into the request:

   - FID of parent directory: FID of "."
   - name to look for: "D"
   - flag indicating that subtree lock is desired

MD server:

- locks the parent directory
- looks for name D
- gets FID of D
- if D is root of subtree lock of other client
  - send BAST to subtree lock holder and wait for response
  - get ordinary lock on D
- MDS returns to the client
  - attributes of object D
  - lock on object D
  - if subtree lock was not canceled
    - set a flag saying that lookup has to continue with ordinary locking

4.8.1 Note

When on downward lookup from the filesystem root a client encounters a subtree lock held by another client, first client has to lookup through STL-ed namespace with ordinary locking assuming that STL holder cached all objects. If the first client worked under its own STL it has to switch to ordinary locking on traversing outsider's STL.

5 Logic specification

5.1 LDLM change

As long as inodebits lock is used solely for locking metadata and as long as subtree lock serves metadata locking as well, it might make sense to introduce subtree lock as an extension to inodebit lock. New bit is added. Then in order to set a subtree lock on an inode locks on all bits are to be acquired.

5.2 Client cache organization

Each object cached under subtree lock on a client has a pointer to the subtree lock.

There has to be a way to iterate over all objects of any subtree below root of subtree lock.

5.3 File system operations

- mkdir, symlink, open(O_CREAT), mknod
  - these all start of lookup in which subtree lock for directory is requested if possible according to client policy and it can be granted according to server policy.
- open
  - if a file has to be opened on MDS, open RPC is sent to MDS
  - MDS checks whether the file is open already
    - if file is not open yet, the client is granted with IBIT lock, with that lock the client may avoid extent locking
    - if file is open by one client only, IBIT lock has to be canceled and futher accesses use extent locking
    - if file is open by more that one client, the client gets open file and has to access the file via extent locking
- rename

  On rename ordinary locks are acquired on objects which are to be modified. Renaming under subtree lock raises the issue of updating cache of objects cached under subtree locks.
  - Rename within one subtree lock
    - Client C1 holds a subtree lock on /a/b/c

    - client C1 does rename("/a/b/c/d1/e1", "/a/b/c/d2/e/f");
      Subtree lock cache is updated to match the namespace change. If subtree lock cache has tree structure similar to dcache parent pointer of "e1" is to be set to "f".

- Rename between 2 subtree locks
    - Client C1 holds two subtree locks on /a/b/c/ and on /d/e/f
    - client C1 does rename("/a/b/c/d", "/e/f/g");
      Two subtree lock caches are modified: "d" and all object below are moved under subtree lock on /e/f/g

- Move out of subtree lock
    - Client C1 holds a subtree lock on /a/b/c
    - client C1 does rename("/a/b/c/d", "/e/f");
      New subtree lock is acquired on /e/f/d. "d" and all objects below are moved under that subtree lock.

5.4 Policy

5.4.1 Client policy

When a client receives MDS's reply for lookup request - the client sets locking state in the inode of lookuped object.

Possible states are:

- inode is protected by subtree lock held on the client
- inode is under subtree lock held on other client, and is locked by ordinary lock
- inode is locked by ordinary lock

On further lookups the locking state of parent directory is used in order to decide, which kind of lock to request for the lookuped object. See 4.6.1 for more details.

On rename ordinary locks are requested for involved objects (if there are no subtree locks already).

5.4.2 Server policy

Subtree lock can not be set for:

- not directories
- root directory

- open directories

For other objects the decision about granting subtree lock is made based upon a statistics collected for each inode on MD server.

The following statistics are collected for each inode on MDS:

For each object we have:

```
/* identifier of client, which accessed the object last */

unsigned long long last_access_origin;
```

For each subtree lock root:

```
/* number of accesses to objects within subtree performed by subtree
lock holder */

unsigned long long stl_holder_access_nr;

/* number of accesses to objects within subtree performed by all clients
but subtree lock holder */

unsigned long long stl_others_access_nr;
```

Having these statistics would allow us to figure out when subtree lock is set not very well and when it may make sense to grant subtree lock:

- when a client accesses root of subtree lock owned by another client, the MDS compares `stl_holder_access_nr` and `stl_alien_access_nr` of the object. If ratio of these counters reached some threshold - it might make sense to cansel the subtree lock
- when a client requests a subtree lock for an object MDS checks `last_access_origin`. The subtree lock is granted if this client accessed the object last or when there were no accesses to the object.

That is supposed to assist to set subtree locks such that most of accesses to objects within subtree locks will be from subtree lock holders.

5.5 Lock conflicts

Lock conflicts get resolved using blocking ASTs. MD server always knows which client to sent a BAST to. In case of sending the BAST to subtree lock holder, it is possible to specify directly which object is in conflict. That would allow a subtree lock holder to choose appropriate reaction. List of possibilities is in 3.6.3.

5.6 Revalidation details

A client C1 holds ordinary lock on an object O1 (chdir(/a/b/c/d/e), O1 is inode of /a/b/c/d/e). C1 is idle for some time and other clients did something with O1 (ls -l /a/b/c/d/e, for example) and C1 loses the lock on O1. Now C1 decides to continue and calls stat("."). Even though C1 lost the lock on O1, it still has means to identify the object O1 on MD server.

1. C1 determines MD server holding the object O1 and sends lock request to the MDS.
2. MDS looks for the object O1.
   - If it is locked by client C2
     - MDS sends a BAST to C2
     - C2 cancels the lock
   - MDS is about to grant the O1 lock to C1.
     - if O1 is root of STL
       - send BAST to STL holder and wait for its response
       - grant C1 with a lock on O1, return value is "STL has been revoked"
     - if O1 is root of the filesystem
       - grant C1 with a lock on O1, return value is "FS root has been reached"
     - grant C1 with a lock on O1, return value is "continue upward traverse"
3. C1 has a lock on object O1.
   - if MDS returned "STL has been revoked"
     - check the lock on object in question, if it is revoked, repeat from the beginning
     - reread the object in question from its MDS
     - revalidation is done, return SUCCESS
   - if MDS returned "FS root has been reached"
     - check the lock on object in question, if it is revoked, repeat from the beginning
     - revalidation is done, return SUCCESS
   - if MDS returned "continue upward traverse"
     - find ".." entry of object O1
     - O1 = parent of O1
     - goto step 1

- Note: The above revalidation algorithm assumes CMD case, for the case of single MDS the revalidation can be performed on the MDS without the exchange by RPCs with a client

6 State management

6.1 State invariants

6.2 Scalability & performance

Subtree locks are supposed to improve performance and scalability in usage pattern where each client work in its isolated namespace and to not make performance worse in all other usage patterns.

6.3 Recovery changes

None.

6.4 Locking changes

Substantial. See above.

6.5 Disk format changes

None.

6.6 Wire format changes

6.7 Protocol changes

6.8 API changes

6.9 RPCs order changes

7 Alternatives

7.1 Revalidation can be avoided

The revalidation would not be required if for all objects to which clients may have direct access to ordinary locking schema were always used.

So, if on lock cancelling for whatever reason (BAST or voluntary drop) we can determine whether the protected object can be accessed directly - we have to request ordinary locking for that object. An object can be directly accessed from several clients (a

directory can be CWD for several clients), so MDS has to maintain a special counter on the lock.

Using NULL lock type of DLM locks seems suitable for this purpose.

Directly accessible objects are objects composing current working directories.

Unfortunately, it looks too complex to maintain set of directly accessible objects in case of cross directory renames. For example, if one client has CWD /a/b/c/d/e/f, and another client does rename("/a/b/c", /1/2/3/4/5/6"), then elements on path /1/2/3/4/5/6 become directly accessible objects and elements of /a/b/ stop being directly accessible objects.

7.2 Lock revalidation and getcwd

Peter Braam noticed that the problem linux getcwd syscall has with network filesystems (lustre, NFS) is similar to lock revalidation problem and he proposed to introduce additional dentry operations which might be used by both getcwd and lock revalidation.

8 Focus for inspections