GNS HLD

Peter Braam & Yury Umanets

20th January 2005

1 *Requirements from the Engineering Requirements Specification (formerly Architecture).*

In this work we are introducing GNS (Global NameSpace) in llite. It is required to do the following:

- store and manage *mount objects* in lustre. Mount objects should store mount related information, which is used for automatically mounting *any* filesystem on the mount object directory in the Lustre filesystem tree. Thus, make lustre operating as a namespace which contains automatically mounting directories. This can unify the namespaces of many Lustre servers and other file systems (e.g. a local disk /tmp can be mounted automatically inside Lustre).
- Architectural decisions were made that:
    1. mount objects would be setuid directories.
    2. these directories would become mount points of completely ordinary file systems
- Automatic mounting proceeds only when truly entering a directory, ie. when doing readdir or lookups inside it, not when doing a getattr on the mount object directory. The latter return a "fake" approximately correct answer.
- Automatically umount filesystems when they are not used for some time. Also automatically unmount all namespace mountpoints on umount.
- To accomodate flexibly mounting ANY file system it was decided to call a wrapper around /bin/mount to mount new file systems on the mount objects.

The architecture was based on a detailed study of existing mechanisms in AFS, Windows, autofs4, eliminating all known disadvantages.

2 Functional specification.

In order to achieve requirements llite is expected to perform the following:

Mount objects (data, not a method!) Directories marked by suid bit as a potential *mount points*. Such directories are mount objects if they contain a special file *.mntinfo*.

Changes to lustre lookup path: Llite should recognize and parse *mount objects, when entering them.* It initiates a mount and wait for completion or timeout when it finds a mount object. Stat calls on the mount objects do not cause mounts. Upcalls to mount should be security concious in using the arguments in the .mntinfo file: only completely trusted answers should be used. Perhaps the mount program should run as a pathname traversing user, to avoid root compromises. Failed mounts or ones that do not complete should be handled gracefully.

Monitor mount objects: A thread monitors all mounted mount objects with particular timeout if they are still in use. If they are not used for some time it umounts them. Also unmount all mounted filesystems at llite module unload time.

Mount completion indication: The user space wrapper indicates completion with an error code of a mount operation performed.

No changes to APIs and protocol are needed.

3 Use cases.

The following use cases can form the basis of the build-and-integration test cases.

1. Mount objects are created as follows:
   (a) create a directory
   (b) write a file .mntinfo in the directory
   (c) chmod the directory u+S
2. Modify a mount object
   (a) chmod u-S on the directory
   (b) edit the file
   (c) chmod u+S
3. Let ../mo be a directory that is a mount object. Path name traversal triggers mounts in the following cases:

   (a) open/readdir/getdents on mo
   (b) operating on any pathname "mo/foo..." (lookup, open, creation inside)

4. As in 3, a mount is NOT triggered by:
   (a) stat mo
5. Unmounts are triggered in reverse order of mounting when:
   (a) directories are un-used.
   (b) umount of the Lustre file system containing the mount points is called
   (c) note that (b) may have to be function recursively.

4 Logic specification.

The following components make up the implementation and should see a detailed design:

Lookup code path Should see modifications to detect and interpret mount objects. Llite stores mount objects as regular files with special names. It should "chdir" to the mount directory and calls user space helper with the mount arguments (in a structured format) contained in the content of *the mount-object/.mntinfo* file. User space helper is expected to mount the filesystem on ".". The GNS code should wait for mount completion and register new mount in GNS thread structures to make it available for GNS thread control.

Monitoring daemon: The timer with specified timeout should be started. Timer callback function should inform GNS thread that it is time to check all mounts and perform umount if needed.

Mount upcall and completion A wrapper around /bin/mount and a completion ioctl on the file system root can be used.

Adding GNS does not change performance or scalability of lustre. No side benefits are recognized.

5 State management.

There are several resources involved in relation to the mount objects:

active, i.e. mounted mount objects mounts performed by GNS should be umounted on timeout (if not used) or in llite module unload time. They should not be leaked by GNS, as system itself will not umount them and they should not lead to failed Lustre unmounts. The controlling thread should be started in llite module load time and stopped in llite module unload time.

in progress mounts Should be handled with a waitqueue and a cookie to wake up the right waiting thread.

strings The mnt file contains a string, which should be highly structured, such as XML in order to enable eas and secure construction of a mount command from this string.

threads performing mounts A concurrency study should be made when multiple threads attempt to traverse a mount point simultaneously.

mount points Generally, mounted directories in a Lustre file system are Unix mount points. They are subject to constraints, such as non-removal by other nodes (where they may not be mounted for one reason or another). For this there is a pinning design and code available which has not been finished. This will be required in a productized version of GNS but can be ignored for cmd2 acceptance tests.

There are no changes to disk format. All the changes are dome in one module - llite.

As to recovery should be decied what to do with mounts on client eviction. There are two possibilities:

- umount all mount points.
- do not umount mount points.

The reason is why we may want to umount them is on eviction all locks are getting canceled and dentries are marked invalid (dentry for mount point too).

6 Architectural alternatives (do not really belong in HLD)

There are three things, which may be implemented another way. They are the following:

- Current implementation makes GNS mount function read *mount object* content and passes it as string to user space helper program . This makes GNS code slightly more complicated than it could be due to needs to manage page of data read from *mount object* file, etc. It would be more sane to pass *mount object* file name to user space helper and make it this way to take care what to do with it. This will make GNS mount stuff slightly simpler. File access in user space is much simpler than in kernel. **This is rejected, because accessing this file causes traversal of the mountpoint and changing its mode bits causes RPC's that affect the mount point on other nodes.**

- Current implementation of GNS mount function involves state sharing between kernel and user space. Sharing state is not good thing in principle, as it adds complexity and assumes bug prone implementations, but sharing state between kernel and user space is totally wrong. Kernel cannot be relying onto user space, kernel does not believe to user space. In this particular case, user space helper program is expected to call special mount completion ioctl() after filesystem is mounted, to let GNS code process further and register successful mount in GNS thread accessible structures to let thread to control it. Here lustre is vulnerable by possible hostile user space helper program, which will not call mount completion ioctl() and will cause lustre hanging. One of alternatives of current implementation could be do not wait for user space signal and check if mount is finished in a loop with sleeping and waiting for mount completion (**YES, please do this)**. Another alternative would be to wait for user space helper process finish, what may mean that mount is performed and we can check the result. There are also another possibilities to avoid this not really needed state sharing. **The remainder of in kernel constructions is rejected because the complexity of the mount program is far too large to simulate in the kernel - mount programs can detect disk layout, make RPC's and perform numerous other complicated tasks.**

- GNS code contains few things which are currently hard-coded and may be replaced by /proc tun-ables. They are the following: mount timeout, GNS timer tick value, user space helper program path and *mount objects* file name. **Good idea, but ONLY use standard proc interfaces.**

7 Focus in inspection.

An inspector should be focused on the following possible issues:

- possible leaks of mounts performed by GNS code.
- GNS thread should always easily be stopped and does not make a kind of hanging in umount time.
- GNS mount should not be performed for directories not marked by suid bit.
- Attempts to concurrently mount a mount object should be analyzed.