Porting Lustre to Operating Systems other than Linux

Ken Hornstein
US Naval Research Laboratory
April 16, 2010
Motivation

○ We do a lot of data visualization on Lustre data, and would like to do that on the Macintosh platform.

○ General strategy of providing uniform access across our entire system.

○ Having Lustre available for more client systems increases Lustre use and visibility.

○ Porting Lustre to one vnode-based operating system would help it be ported to another.
History

- Initial Lustre port done in 2005.

- Targeted toward MacOS 10.3 (Panther).

- Some of the design decisions made for that port made long-term support difficult.

- Port reached some level of functionality, but has been bit-rotting for a number of years.
Fast-Forward to the Present

- Current tree does not contain all of the port.

- libcfs has been split off to OS-specific directories and has a set of Darwin (MacOS X) functions and header files.

- A fair number of modules (obdclass, Inet, ksockInd) had kernel module metadata files (Info.plist).

- Some of the include files under lustre/include have been segmented off with OS-specific versions.

- Build system has knowledge of MacOS X.
But ...

- Lustre has been moving forward for five years.

- No effort had been made on maintaining cross-platform portability.

- The port uses many old MacOS X kernel interfaces.

- Missing bits of port contain some of the more important pieces (the page caching code and the vnode interface).
Challenges

- Lack of documentation of Lustre internals.
  - Understanding Lustre Filesystems Internals helps.

- Lack of documentation of Linux internals.

- Lack of documentation on MacOS X internals.
Design Decisions

- Try to concentrate MacOS X changes to libcfs and OS-specific files.

- Minimize #ifdefs in generic Lustre code.

- No changes at all to MacOS X.

- Target kernel modules instead of FUSE.

- Base code on master branch.
Initial Work (2-4 weeks)

- Work through bitrotted code in libcfs.

- Many Linux interfaces prefixed with "cfs" - rename or implement functionality.

- Some code actually simplified (kernel thread argument handling & timers)

- Switched many interfaces (mostly locking) to newer interfaces.
  - Spinlocks - IOSimpleLock
  - Mutexes - IOLock
  - Semaphores - IOResursiveLock
Initial Work (continued)

- libcfs networking code cleaned up (much simpler!).
- Lustre tracefile implementation problematic (CPU numbering).
- Lots of challenges with ioctl interface (32 bits versus 64 bits).
- Kernel-userspace communication switched to using socketpair().
- Ported ptlctl to test basic networking functionality.
Next Steps (6-7 weeks)

obdclass took the largest amount of effort.

- All modules call through it (register callback interfaces that are used by all other modules).

- Contains the cache handling code (cl), llog, encryption/checksum interface, part of VFS interface, sysctl handling, inode attribute management, capability management.

- Significant parts of obdclass are OS-specific!
Next Steps, continued

- Fair amount of changes were required to simply switch to cfs prefix for functions/datatypes (struct page -> cfs_page_t).
- Switching away from static lock initializers to explicit lock allocation/free in module startup and shutdown.
- Segregate Linux-specific functions into files in "linux" directory.
- Write MacOS X versions of Linux functions (crypto interface) and bring over missing functionality from Linux (radix tree).
Next Steps, continued

- Once obdclass was ported, ptlrpc was next.

- ptlrpc work exposed a number of bugs in the MacOS X versions of the Linux synchronization functions (mostly completion and waitq).

- After ptlrpc was done, the rest of the modules went relatively smoothly.

- Remaining module work consisted of switching away from Linux include files and #ifdef’ing out procfs support.
Crossing the Finish Line (3 weeks)

- llite is the module that interfaces with the Linux VFS system. By necessity it is very Linux-specific.

- A direct port of llite would have resulted in a gigantic number of #ifdef’s and massive restructuring, and the result would unlikely ever be accepted back into Lustre.

- Decided to create a new module to handle the MacOS X vnode interface (lvnode).
An Aside about Vnodes

- Interface developed by Sun as part of development of NFS.
- Vnodes are virtual versions of inodes; one vnode per filesystem object (files and directories). In MacOS X the vnode is an anonymous structure (cannot access contents).
- Filesystems create vnodes as necessary (when files are looked up by the operating system) and fill in filesystem-specific information in the vnode private area.
- A filesystem provides methods at vnode creation time to perform operations on the vnode (such as create, read, write, unlink).
Lvnode Implementation Details

- Lvnode indexes Lustre files via the fid (unique identifier per filesystem).

- Vnode contains pointer to Inode structure, which contains fid, mount point (our version of superblock), which in turn contains pointer to our metadata and data exports.

- The operating system manages the caching between names and vnodes (and due to vnode containing Inode, the mapping between names and Lustre fid).
More Implementation Details

- The data flow in llite due to Lustre caching is ... confusing. Also, not sure how to interface it with the MacOS X buffer cache.

- For the first effort, decided to skip caching completely.

- Since there were problems in my first attempt to use intent locks, attribute caching is not implemented at this time as well.

- Readdir performance is sub-optimal; also, no statahead/readahead.
Challenges During Implementation

- Misuse of intent locking caused LBUG() on MDS!

- Low level differences between Linux and MacOS X manifest at a high level (bit ordering difference caused failure reading config log).

- Memory management of Lustre API not documented anywhere.

- Lack of communication between client and server results in client eviction; solution is to use the pinger, but that seems wrong.
The Ugly Details

- Currently open/close are not actually registered on the MDS.

- readdir() calls md_readpage() for each call.

- I/O is done via obd_brw() (one or more per each read/write() call), and is done synchronously.

- setattr currently not supported (although looks relatively straightforward).
Unanswered Questions

○ For caching, should we use Lustre’s caching (which seems to be designed to interface with the Linux VM system), or use the operating system’s buffer cache?

○ MacOS X does not have anything like the Linux shrinker, so there is no way to know if VM pressure is an issue.

○ What work is necessary to cooperate with the MacOS X Finder?
Future Work

- Clean up resource leaking (lock leaks are terrible, due to lack of lock cleanup needed on Linux).

- Implement data caching!

- Implement intent locking to cache attribute and file data.

- Implement Kerberos support.

- Implement Infiniband support (o2iblnd).
Things That Would Aid Portability

- Greater discipline on using "cfs" prefix in generic Lustre code.

- Break up OS-specific obdclass parts into different directory, or even a different module.

- Purge use of struct inode and struct super_block in obdclass (using cfs_inode_t and cfs_super_block would be fine).

- Work on creating a more generic cache system to interface with buffer caches used by other operating systems.
Long Term Plans

- We get funding for doing new things; developing MacOS X port is something new, but long-term support for a MacOS X client is NOT new work.

- Would like to eventually host the source code on the Oracle git server.

- In a perfect world, MacOS X port would be supported by Oracle (pipe dream!) or by the community, and would be considered a supported client platform.
Any Questions?