# CITI - ARC Joint Project Status Report

## Center for Information Technology Integration

## January 10, 2008

This memorandum reports on the status of a joint project between IBM, Almaden and CITI, University of Michigan under Agreement No. A0550295. The Statement of Work specifies three major tasks:

- Locking, delegations, and shares for cluster file systems
- Replication and migration
- pNFS

CITI led or participated in all of the work reported work below, in coordination with IBM developers. The principal mechanism for coordination is a weekly conference call among developers at CITI, IBM, Network Appliance, Panasas, Red Hat, SGI, Google, and EMC.

## 1 Locking, delegations, and shares

| Task 1.1 |
| --- |
| • Continue to develop and test fair queuing for byte-range locks.<br>• Respond to feedback from Linux developers and maintainers.<br>• Work with Linux maintainers to include a solution for fair queuing in the mainline kernel. |

CITI submitted a set of patches for review by Linux developers and maintainers; see git://linux-nfs.org/~bfields fair-queuing.

This code introduces the notion of a “provisional” lock, which corresponds to a lock request that is anticipated to follow the release or downgrade of another lock. When a lock is released, that lock’s conflict list is examined. If a request remains in conflict with some other lock, it is placed onto the corresponding conflict list. Otherwise, a provisional lock is granted and the corresponding process is awakened. The awakened process then issues the (now conflict-free) request, and the provisional lock is upgraded.

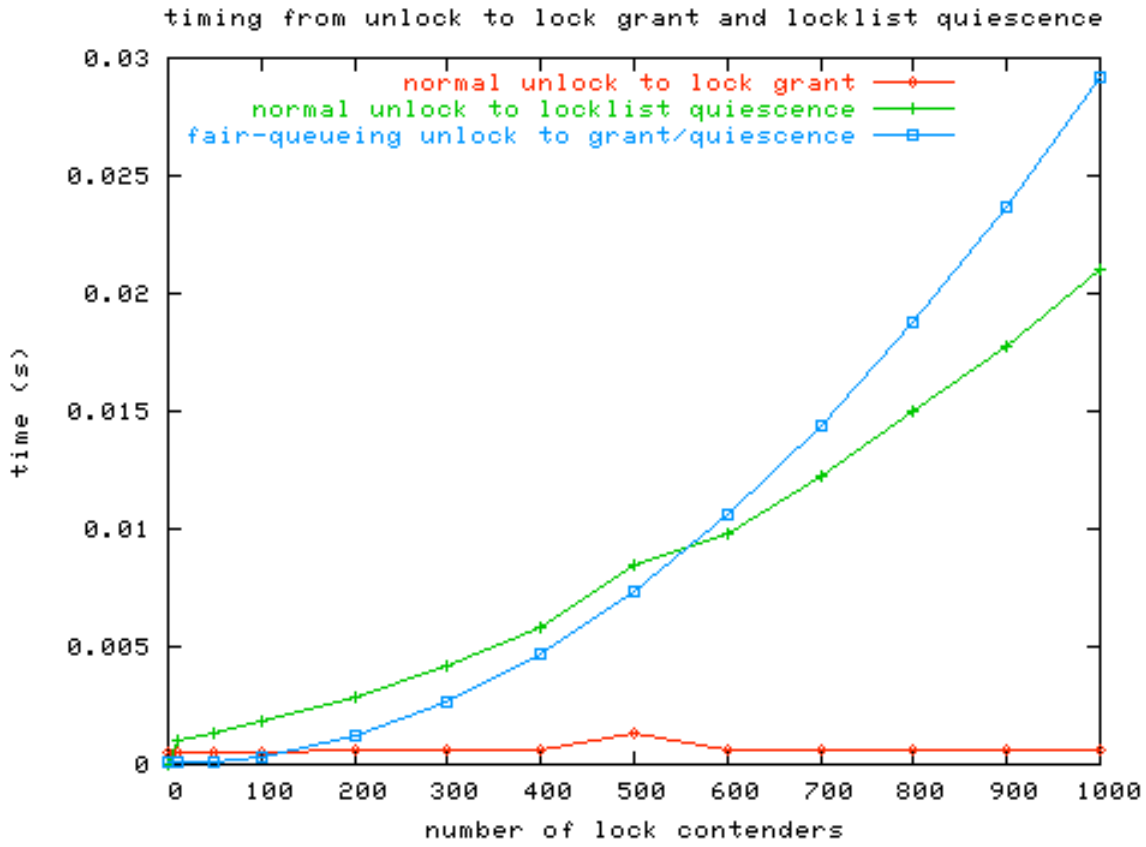
In order to convince Linux developers and maintainers to include this code into the mainline kernel, we need to make a stronger case. A rationale for the fair queuing code might have several dimensions:

- NFSv4 lock starvation

  Because NFSv4 polls for conflicting locks, local (blocking) lock requests have a natural advantage. The patches would clearly help prevent NFSv4 lock requests from being starved by local lockers (or by other NFSv4 clients with more frequent polling), but the Linux developers and maintainers need to see workloads demonstrating that starvation happens in practice. We do not have workloads like that; indeed, in specifying a polling interface, the NFSv4 protocol implicitly assumes that lock conflicts are rare, so such workloads may not exist.

- Local lock performance

  The use of provisional locks can improve local lock performance by avoiding the “thundering herd” problem. However, some (quick and dirty) testing did not indicate improvement:




Although we have not delved too deeply into explaining the performance of local lock contention, we surmise that the quadratic behavior is due to the POSIX requirement for deadlock detection. However, the experiment does not help to build a case for performance improvement.

- Reduce code complexity

  Provisional locks might help simplify lock support in the Linux kernel and make it easier to extend the lock API to support asynchronous calls, but our sense is that provisional locks actually complicate matters. Furthermore, we figured out a way to support asynchronous requests without requiring provisional locks.

- Improve local lock fairness

  POSIX requires that locks be granted as soon as possible, so most heuristics to improve fairness (e.g., delaying read locks to prevent starvation of writers) are prohibited. Consequently, any improvement in fairness is likely to be subtle.

In summary, it is difficult to make a strong case to Linux developers and maintainers for fair queuing support in the mainline kernel.

| Task 1.2 |
| --- |
| • Design and implement an interface for NFSv4 open share modes. |

We made no progress in building a case for open share mode support with Linux developers and maintainers. The obstacles we face remain unchanged from a year ago: DENY shares are alien to POSIX and Linux.

This is not perceived by Linux developers and maintainers to be a problem: Linux is incapable of

issuing DENY access, so the absence of support for DENY access in Linux file systems does not meet the NFSv4 access needs of Linux clients, just Windows clients. Furthermore, not even off-the-shelf Windows clients benefit, as NFSv4 for Windows is a third-party add-on (from Hummingbird).

| Task 1.3 |
| --- |
| • Implement and test the proposed cluster-friendly interface for file delegations. |

The basic lease() interface described in the Linux NFS wiki

http://wiki.linux-nfs.org/wiki/index.php/Main_Page

(see cluster coherent NFSv4 and delegations under Cluster Coherent NFS) is in the mainline kernel. Marc Eshel has tested with GPFS. GFS2 does not support cluster-wide leases, so the interface has not been tested there. We have corresponded with OCFS2 and CIFS developers, but we are unaware of any testing there.

## 2 Replication and migration

| Task 2.1 |
| --- |
| • Implement and test applications that support server-to-server state transfer. |

Migration of client state from one server to another builds on and extends existing mechanisms for reboot recovery with a state transfer RPC service and a command line utility that sends RPCs to listeners on the source and target servers. We are in the process of documenting this work in the Linux NFS wiki; see cluster client migration prototype.

| Task 2.2 |
| --- |
| • Design and implement a client notification mechanism to inform migrating clients that state established with the initial server is valid on the target server. |

The Linux NFSv4 client reacts to an NFS4ERR_MOVED error by attempting to use its existing CLIENTID with the target server. This obviates the need for a client notification mechanism.

| Task 2.3 |
| --- |
| • Design and implement client support for volatile file handles, file handle recovery on expiration, following of referrals, and state recovery or expiration on migration.<br>• Design and implement triggered generation of server migration and referral events. |

The Linux NFSv4 client follows referrals appropriately when it receives an NFS4ERR_MOVED notification.

Client state housekeeping has been re-organized so that per-server state is easily identified, allowing recovery or expiration on migration.

The combination of Tasks 2.2 and 2.4 provides triggered generation of server migration and referral events.

The Linux NFSv4 client does not support volatile file handles or their recovery, so CITI is unable to deliver migration support for volatile file handles.

| Task 2.4 |
| --- |
| • Implement and test a server interface to FS_LOCATIONS information for a given file system that uses the exports file and LDAP. |

Support for FS_LOCATIONS information in the exports file and LDAP is in the mainline kernel. The CITI LDAP schema is being used as a basis for the NFSv4 Federated-FS effort.

# 3 pNFS

| Task 3.1 |
| --- |
| • Continue to implement and test NFSv4.1 features, including sessions, RECLAIM_COMPLETE, the generic pNFS client and server, and the file layout driver.<br>• Respond to IETF and Linux kernel developer comments |

We made many contributions to the specification and implementation of pNFS.

- We rebased from the 2.6.17 kernel to 2.6.18.3. This was done largely through a CVS rebase, but also involved hand coding the I/O path.
- We helped write and debug the sessions code for the 2.6.18.3 kernel.
- We helped write and debug the file layout I/O path for the 2.6.18.3 kernel.
- We helped rebase pNFS to draft 13.
- In November 2006, we tested pNFS scalability properties with a pNFS client located at CITI accessing a GPFS cluster located at the School of Information, several miles distant (250 μsec RTT). The client is equipped a 10 Gpbs interface that connects to the University's 10 Gbps MAN. Cluster nodes use a 1 Gbps interface attached to a Force10 S50 switch on the MAN.

  Performance testing verified that increasing the number of pNFS stripes yields an increase in READ and WRITE throughput:

  - READ throughput with one stripe yields .75 Gbps; READ throughput with five stripes yields 3.6 Gbps, a 4.8× improvement
  - WRITE throughput with one stripe yields .43 Gbps; READ throughput with five stripes yields 1.5 Gbps, a 3.5× improvement

  In anticipation of addressing pNFS WAN scalability, we tested NFSv4.1 WAN scalability in September 2007 with a 10 Gbps client at CERN and a 10 Gbps server at CITI (120 msec RTT) using a 10 Gbps path over UltraLight. These tests indicate the need for further kernel tuning. Initial tests yielded abysmal performance: 114 Mbps for TCP and 3.2 Mbps for NFS I/O. Tuning IP (routing and MTU) and TCP (congestion and buffer management) achieved 7 Gbps raw TCP throughput. NFS READ performance was measured as high as 2.8 Gbps, but repeatable performance was still only 120 Mbps. WRITE performance was much worse, 6.4 Mbps, due to TCP window problems on the NFSv4.1 server. This work is ongoing.
- We organized and hosted the October 2007 NFSv4.1 bake-a-thon.

  We rewrote the file layout device implementation for the pNFS client and server at the bake-a-thon to interoperate with the Solaris pNFS client and server.

  Interoperability testing at the October 2007 bake-a-thon yielded the following:

  - File layout: full interoperability among the Solaris client and server, the Linux client, the NetApp Linux-based spNFS server, and the IBM GPFS server
  - Object layout: full interoperability between the Linux client and the Panasas server. We demonstrated the ability to copy from object layout to file layout on a single Linux pNFS client.
  - Block layout: READ interoperability between the Linux client and the EMC server.
- We contributed to the development of the pNFS specification and attended IETF 67 and 68.

We did not implement RECLAIM_COMPLETE (yet)