# How the *Global File System* uses Dlock


Kenneth W. Preslan

Parallel Computer Systems Laboratory
University of Minnesota
http://gfs.lcse.umn.edu


September 16, 1998

# Outline

1. Short Description of GFS

2. Why Dlocks?

3. Current GFS

   - Locking
   - Caching
   - Crash Recovery

4. New Features

   - Dlock Timeouts
   - World Wide Names
   - Shared Locks

5. Summary

# What is GFS?

- GFS is a Shared-Disk filesystem for Fibre Channel

- GFS is Symmetric (No File Manager)

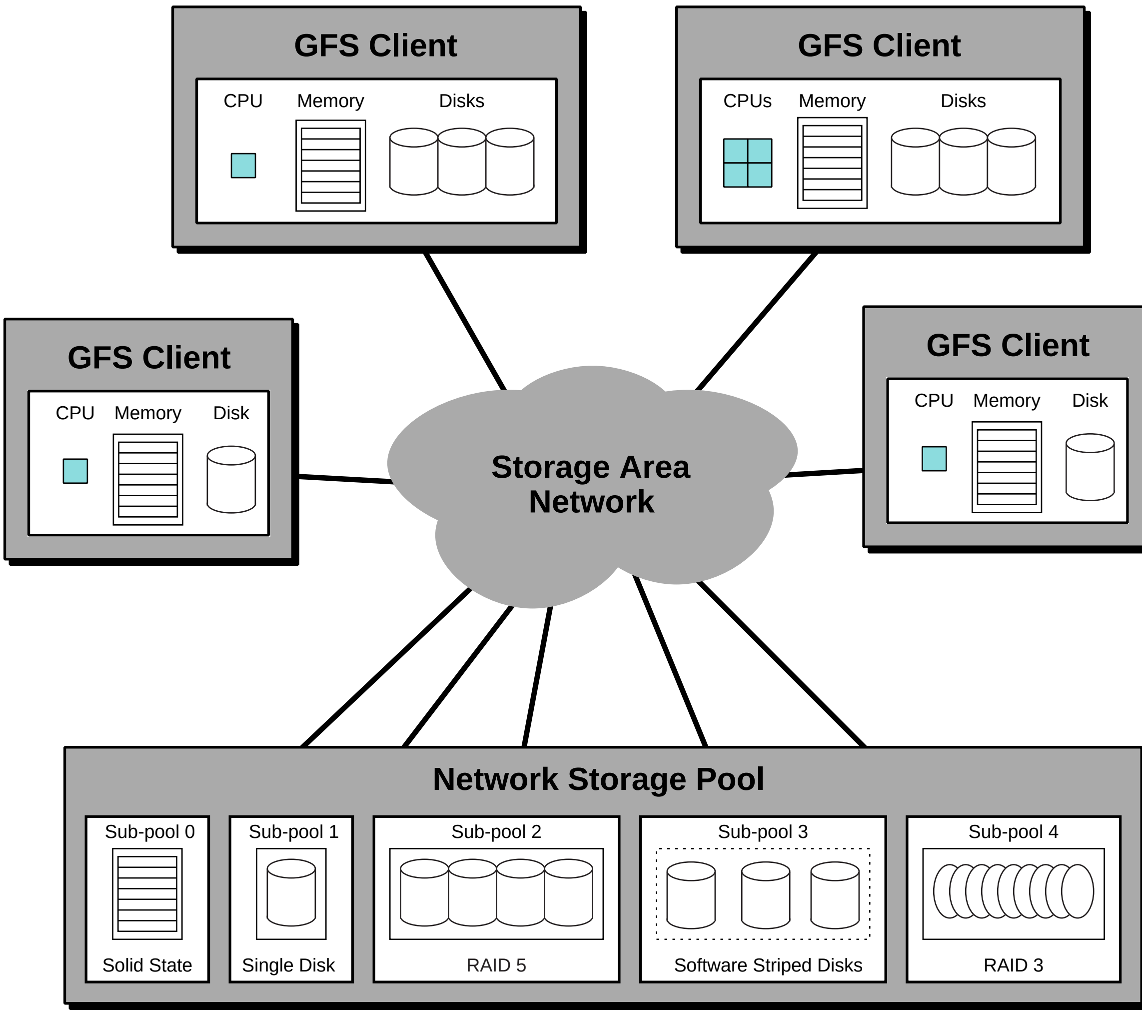
- Currently implemented on Irix and Linux

- Two parts

  1. The Network Storage Pool Driver

  2. The File System

- Uses an earlier implementation of Dlock on Seagate drives


GFS Client
CPU
Memory
Disks
GFS Client
CPUs
Memory
Disks
GFS Client
CPU
Memory
Disk
GFS Client
CPU
Memory
Disk
Storage Area
Network
Network Storage Pool
Sub-pool 0
Solid State
Sub-pool 1
Single Disk
Sub-pool 2
RAID 5
Sub-pool 3
Software Striped Disks
Sub-pool 4
RAID 3

# Pool

- Provides an uniform address space of data blocks and Dlocks to the FS.

- Takes lock and unlock commands from the FS and translates them into SCSI Dlock commands.

- Handles retries if the lock is busy, lock time-outs, and lock resets

# The File System

- Very similar to a local filesystem, but with locks for consistency

- Each client is in charge of keeping the meta-data consistent as it writes data

- Handles deadlock avoidance, caching, crash recovery

# Why Dlocks?

- SCSI devices are more reliable than most hosts

- Locks can be distributed across many disks without the complexity of DLMs

- Dlocks are faster in the firmware of a drive than on a general purpose computer

- Dlocks aren't connected to any particular part of the medium. A Dlock on one device can be used to lock data blocks on another device. What each lock represents it totally up to the initiators using it.

# Current GFS

- Uses a early version of the Dlock Spec – no shared locks, lock timeout, or World Wide Names

- Implemented on Irix and Linux

- GNU GPL source on http://gfs.lcse.umn.edu

# Locking

- Dlocks are used to perform mutual exclusion when metadata is updated.

- One Lock for the Superblock, Root Inode, and Rename Lock

- One Lock for each *Resource Group* (bitmaps)

- Each Inode hashes to a Dlock – Many to one mapping

- Dlocks are only held for short read-modify-write operations.

# Caching

- Dlocks provide a means of validating client-held caches of data

- Two types of Unlock actions

  1. Unlock – Regular unlock

  2. Unlock Increment – Increment the lock's *Version Number*

- Operations that change metadata on disk use Unlock Increment

- Read-only operations use Unlock

# Caching

- Each Lock action returns the version number for that lock.

- Clients compare the version number returned from a lock action with the version number from when they last held the lock.

- If the two version numbers match, the cache is valid.

# Caching – Example

| | **Machine A** | | **Lock** | **Machine B** | |
|---|---|---|---|---|---|
| | Action | Valid | | Action | Valid |
| 0 | | | U,0 | | |
| 1 | Lock Shared | No | S,0 | | |
| 2 | No Modify | | S,0 | | |
| 3 | Unlock | | U,0 | | |
| 4 | | | S,0 | Lock Shared | No |
| 5 | | | S,0 | No Modify | |
| 6 | | | U,0 | Unlock | |
| 7 | | | E,0 | Lock Exclusive | Yes |
| 8 | | | E,0 | Modify | |
| 9 | | | U,1 | Unlock Incr | |
| 10 | Lock Shared | No | S,1 | | |
| 11 | Modify | | S,1 | | |
| 12 | Unlock Incr | | U,2 | | |
| 13 | | | S,2 | Lock Shared | No |
| 14 | | | S,2 | No Modify | |
| 15 | | | U,2 | Unlock | |
| 16 | Lock Exclusive | Yes | E,2 | | |
| 17 | No Modify | | E,2 | | |
| 18 | Unlock | | U,2 | | |

# Crash Recovery

- We want to be able to detect when a client holding a lock crashes
- Currently, GFS does this by repeatedly polling the lock
- If the version number doesn't change for a set amount of time, it is assumed that a client has crashed.
- The lock is then reset and the FS can invoke a recovery routine.

# Areas of improvement for GFS

- The current GFS implementation works, but there are areas for improvements

- GFS currently doesn't do any write caching.

- Some locks (like the root inode lock) are read a lot, but not written.

- Online recovery is hard

# Dlocks are held longer

- Instead of a Dlock being acquired and released each operation, it is held for a long period of time.

- While the lock is held, write-caching is possible.

- Write-caching makes a journal-ed or log-structured file system efficient.

- The effect of the latency involved in getting a Dlock is lessened.

- But, this breaks our recovery scheme. A Dlock isn't constantly being locked an unlocked, how do we know when it is held by a failed initiator?

- But, if some other client has a lock and is holding it for a long time, how do we get it?

# Dlock Timeouts

- Dlocks time out (and become unlocked) if they are left idle for too long

- All that is required to recover from a failed client is to keep retrying the lock until it succeeds.

- When a lock action succeeds, a flag is passed back letting the client know whether or not the lock was expired.

- A *touch lock* command exists that lets a client reset the timeout on one of its locks.

- A client failure on a infrequently used Dlock is automatically detected

# World Wide Names

- When a lock command fails, a list of World Wide Names is returned to the client.

- Each WWN belongs to a client that is currently holding the lock

- Allows a client who wants a lock to ask other clients to release it.

- Allows arbitrary crash detection algorithms.

# Reader/Writer Locks

- Locks are acquired in one of two modes: Shared or Exclusive.

- A shared lock can be held by multiple clients that all want to read – but not write data.

- An exclusive lock is held by a client that wants to write.

- Locks that are read a lot, but seldom changed (like the root directory) can be held as shared locks to reduce congestion

# Summary

- GFS has been implemented using a early Dlock Specification

- GFS shows that it is possible to implement a shared-disk filesystem without having to rely on a centralized file manager.

- Future version will efficiently scale from a multiple machine cluster down to a single client.

- Dlocks provide a robust, distributed locking mechanism.