An Overview of The Global File System

David Teigland University of Minnesota
teigland@ece.umn.edu
Ken Preslan Sistina Software
kpreslan@sistina.com
Matthew O'Keefe University of Minnesota
okeefe@ece.umn.edu

http://www.globalfilesystem.org

Outline

- Network Attached Storage, Fibre Channel, and Shared Disk File Systems
- The Global File System
  - –The Network Storage Pool
  - –The File System
  - Structure
  - Features
- Recovery (Journaling)
- Performance
- Future Work

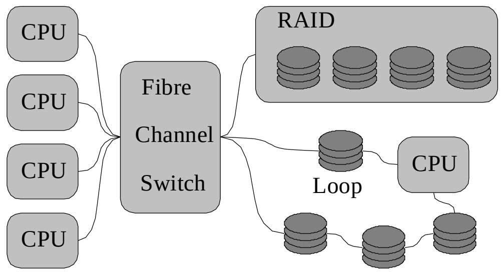
Network Attached Storage

- The power of microcontrollers in disk drives has steadily increased
- They are now powerful enough to manage network connections
- Hence, Network Attached Storage
- New approach to disks − Machines now share disks. They don't own them.
- Storage Area Networks (SANs)

Fibre Channel

- Fibre Channel is a combination of a local area network and a storage bus.
- A Gigabit interface
- Can do both SCSI and IP at the same time
- Point−to−Point, loop, and switched configurations

A Fibre Channel Network

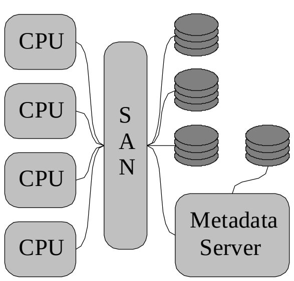


Shared Disk File Systems (SDFS)

- Each machine accesses the disks as if they were local
  - Faster access
  - Greater availability
- Need a method of synchronization
  - 3rd Party Transfer (Asymmetric)
  - Dlocks/GFS (Symmetric)

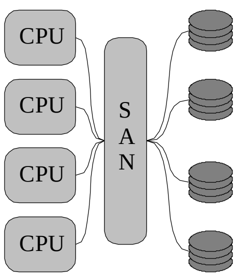
Asymmetric

- Machines share disks containing data, not metadata
- Metadata is controlled by a central server
- The server provides synchronization between clients
- Machines make metadata requests (create, unlink, bmap) to the server
- Machines read actual data from the disks
- Similar to a traditional DFS
- CXFS, DataDirect, MountainGate, Mercury



Symmetric

- Machines share disks containing data and metadata
- Metadata is managed by each machine as it is accessed
- Synchronization is achieved using global locks (Dlocks or a Distributed Lock Manager (DLM))
- A local file system with inter−machine locking
- GFS, VaxCluster, Frangipani



Comparison

- Asymmetric
  - −Simpler to implement
  - −Metadata Server is a Single Point of Failure
  - −Metadata Server is a bottleneck
- Symmetric
  - −Data, metadata, and locks are distributed
  - −No Single Point of Failure
  - −No single device involved in all transactions
  - −No dedicated hardware
  - −Recovery is complex

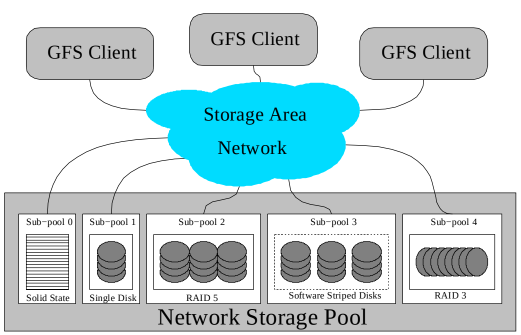
The Global File System

- Symmetric Shared Disk File System
- Open Source (GNU GPL)
- 64−bit Files and File System
- High Performance
- Originally for Irix, now Linux and FreeBSD
- Comprised of two parts
  1. The Network Storage Pool Driver
  2. The File System

The Pool Driver

- A Logical Volume Driver for Network Attached Storage
  - −Combines multiple disks into one logical address space
  - −Combines multiple lock devices into one logical lock space
- Handles disks that change IDs because of network rearrangement
- A Pool is made up of SubPools of devices with similar characteristics

A Network Storage Pool



Volume Driver Layering

- Pool supports striping
- Other RAID levels by layering Pool on MD
- Linux−LVM also benefits from stacking LVM above MD devices
- LVM, MD, and Pool can call *lvm_map*, *md_map* and *pool_map* directly in *ll_rw_blk*
- Or call map function through function pointer

Generic Mapping

- Add *map_fn* function pointer in *blk_dev*
  - Eliminates driver specific code in *ll_rw_blk* (much cleaner and less code)
  - Clean way to make volume driver modular
  - No limit on number or order in which logical devices are stacked
- *dev−>map_fn* is used like *dev−>request_fn*
- *make_request* is handled the same way with *makerq_fn* pointer

Generic Mapping

- New code segment in *ll_rw_block()* replacing code between `#ifdef CONFIG_BLK_DEV_X`

```
tdev = dev;

while (tdev->map_fn) {
     if (tdev->map_fn (bh[i]->b_rdev, &bh[i]->b_rdev,
                       &bh[i]->b_rsector,
                       bh[i]->b_size >> 9)) {
          printk (KERN_ERR "Bad map in ll_rw_block\n");
          goto sorry;
     }
     tdev = blk_dev + MAJOR(bh[i]->b_rdev);
}
```

The File System

- A high performance local file system with inter−machine locking
- Optimized for Network Attached Storage
- When the locks are removed, GFS makes a good local file system
- Two types of locks
  - SCSI Dlocks
  - IP based Locks

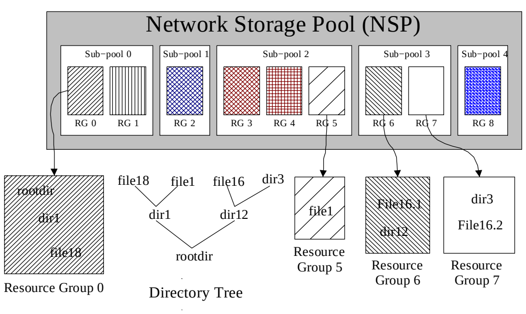
Device Locks

- Global locks that provide the synchronization necessary for a symmetric SDFS
- Lock located on the network attached storage devices
- Accessed with the Dlock SCSI command
- Features
  - Advisory
  - Reader/Writer
  - Version Numbers enable cache coherence
  - Each lock has a list of the machines holding it
  - All locks held by client expire if the client fails to heartbeat the drive

GFS Layout

- A SuperBlock with the location of the resource groups
- Resource Groups
  - Similar to EXT2's Block Groups or XFS's Allocation Groups
  - Bitmaps
  - Blocks (inodes, indirect, data)
  - Each resource group has a number of Dlocks

A GFS File System

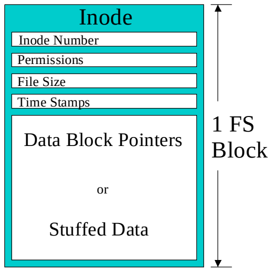


GFS Features

- Dynamic inodes
- Flat/64−bit metadata structure
- Platform independent metadata
- Extendible Hashing Directories
- Full use of the buffer cache
  (full read and write caching)
- Interchangeable Locking Modules

Dynamic Inodes

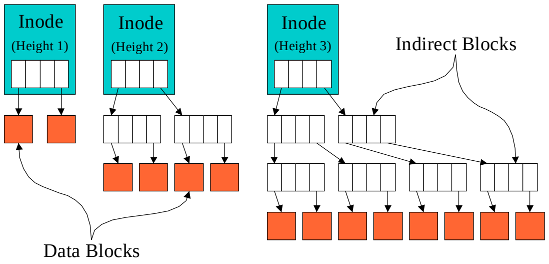
- No preallocated inode tables
- Each inode is just a file system block
- There can be as many inodes as there are file system blocks
- Inode numbers are just disk addresses
- Inodes identified in the allocation bitmaps
- Inodes can be *stuffed* for space efficiency



Flat/64−bit File Structure

- All file sizes, offsets, and block addresses are 64 bit
- File metadata trees are of uniform height
- All direct pointers, or all indirect pointers, or all double indirect pointers...
- Tree height grows to accommodate the size of the file
- No practical file size limit
- Simplifies the block mapping routines

Flat/64−bit File Structure



Platform Independent Metadata

- All on−disk structures are in a platform independent format
- Differences in structure packing are handled
- Differences in endianess are handled
- Very important for GFS because all clients must understand and manipulate the metadata

Fast Directories

- Small directories are stuffed in the inode
- Larger directories use a technique called *Extendible Hashing*
- File names are hashed into keys that are indices into a growable hash table
- Faster than B−Trees
- A bit more space hungry

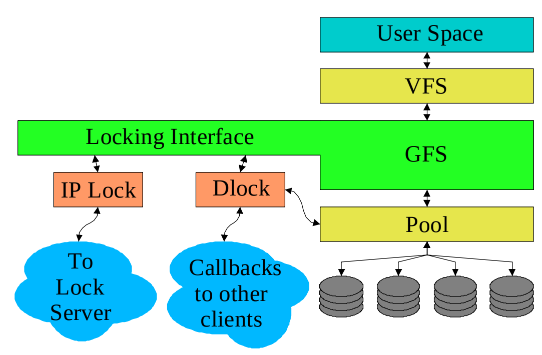
Using the Buffer Cache

- The buffer cache is critical to the performance of a file system
- Linux's buffer cache is written with the assumption that only one machine is modifying the data on the disks
- GFS uses routines to keep track of the buffers in the buffer cache and invalidate them when necessary
- GFS can do both read and write caching

Interchangeable Locking Modules

- Want GFS to be independent of the type of inter−machine locking available
- Created a locking interface to allow modules to plug into GFS
- Each module translates between the locking that GFS expects and the locking available
- The interface allows both very minimal locking protocols and very complex protocols
- Fairly well documented in GFS2/src/fs/gfs_locking.h

Organizational Structure



Registration

- Locking modules register themselves with GFS using the function *register_lock_proto*()
- The module registers a structure containing a structure of operations that the module implements
- Operations: mount, unmount, lock, unlock, release and reset

Operations

- *Mount* − Called once at mount time to set up the lock space
  - Table Name − a name identifying the lock space to be used. (e.g. The Pool name)
  - Call Back − Allows the locking module to ask GFS to unlock a lock
- *Unmount* − Called at unmount time to close the lock space

Operations

- *Lock* − Acquire a Global Lock (Glock)
  - Lock Number
  - Action − Acquire, Try, or Test
  - Flags − Shared, Commute, and Commute_Mod
  - Returns − Held, Shared, Cacheable, Expired, Need_S, and Need_E
- *Unlock* − Unlock a Glock
  - Lock Number
  - Flags − Modified

Currently Implemented Protocols

- Nolock − Dummy locks for local file systems
- Dlock−0.6 − Old lock specification (Exclusive locks, Synchronous)
- Dlock−0.9.4 − The 0.9.4 specification (Reader/Writer, Asynchronous)
- Dlip−0.9.5 − The 0.9.5 specification over TCP/IP (drives do not need to support Dlock)
- Future: DLM ?

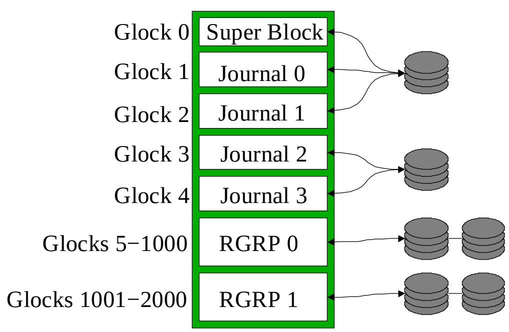
Recovery

- A FSCK is the classic means of recovery after a crash
  - Slow (time proportional to FS size)
  - The file system must be offline
  - Not acceptable for shared disk file systems
  - Now functional for GFS, will be improved
- Journaling solves these problems
  - Recovery time proportional to FS activity
  - Online recovery is possible

Layout for Journaling

- Having multiple clients share a journal is too complex and inefficient
- Each client gets its own journal space
- Each journal space is protected by one lock that is acquired at mount time and released at unmount (or crash) time.
- Each journal can be on its own disk for greater parallelism
- Each journal must be visible to all clients (for recovery)

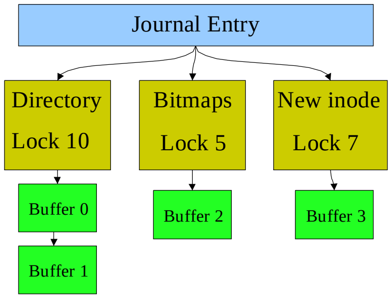
GFS Layout



Journal Entries

- Composed of the metadata blocks changed during that operation (and a header)
- Each entry has one or more Glocks associated with it
  - Standard GFS locks that protect each piece of metadata
  - For instance, a creat() entry would have locks for the directory, the new dinode, and the bitmaps.

A Journal Entry (in memory)



Journaling

- Asynchronous
  - Similar method to XFS
  - Multiple journal entries are cached in−core
  - Entries are committed to disk in groups asynchronously
  - Metadata buffers for a journal entry are pinned in memory (can't be synced) until the entry is committed.
  - When journal write is complete, dirty metadata buffers can be synced

Journaling in GFS

- All journal entries are linked to one or more Glocks
- Before Glock is released for other machine:
  1. Flush journal entries for Glock to log
  2. Sync in−place metadata buffers
  3. Sync in−place data buffers
- Only transactions dependent on the requested Glock need to be flushed (or indirectly dependent)

Journaling in GFS

Journal Entry	Glock # 2	3	6	8
1	X	X		
2		X	X	X
3	X	X		
4			X	X

X represents in−memory metadata buffers which will be written to the journal

- Glock 6 is requested by another machine
  - flush entries 1,2,4 to log in order
  - in−place metadata and data buffers are synced for Glock 6
  - Glock 6 is released

Journaling in GFS

- Initial version will by synchronous to allow work on recovery
  - This is quicker and orthogonal to recovery code
  - Performance will be improved after recovery is in place by moving to async method
  - The journal entry and in−place metadata are synced before locks are released for each operation

Recovery − Initiation

- Journaled recovery is initiated by:
  - mount time check if any journals were shutdown uncleanly
  - locking module reports an expired client when it polls or detects expired machines
  - client tries to acquire Glock and locking module reports it's expired
- In each case, recovery kernel thread is called with expired client's ID
- Machine attempts to begin recovery by trying to acquire journal lock of failed client

Recovery − Failed Clients

- A client which fails to heartbeat its locks but is still alive could do IO while other machines are trying to recover for it.
- Causes filesystem corruption
- Two solutions:
  - Forcably disable failed client (shoot it in head)
  - Fence out all IO from the failed client using Fibre Channel switch
- This is the first step of recovery after acquiring the journal lock of failed client

Recovery of Journal

- Find head and tail of journal entries
- Ignore partially committed entries
- For each entry
  - try to acquire all locks associated with that entry
  - determine whether to replay it and do so if needed
- Mark all expired locks *not expired* for failed client
- Mark the journal as recovered

Replaying Entries

- Decision to replay entry is based on generation number in primary pieces of metadata
  - dinode
  - bitmap headers
- When these are written to log, generation number is incremented
- Replay journal entry if generation numbers in entry are larger than in−place data

Recovery

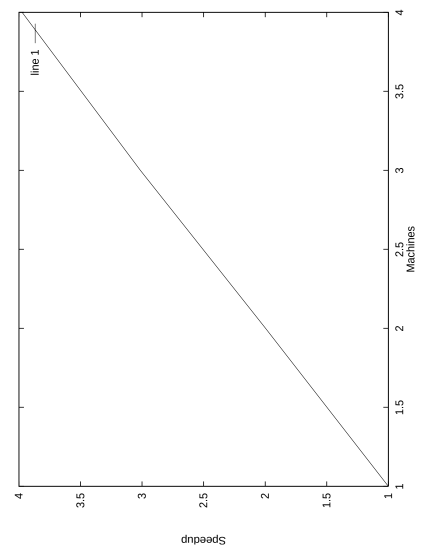
- Machines can continue to work during recovery unless they need a lock which was held by a failed client
- Advantage over FSCK

Performance

- Test configuration
  - 4 Alphas with Linux kernel 2.2.11
    - 21164, 533 Mhz, 128 MB memory
    - Qlogic 2100 FC adapters
  - 4 four−disk JBODS (16 drives)
    - Seagate ST39175FC "Barracuda" 9 GB disks
    - Dlock version 0.9.4
    - Each JBOD is a separate striped subpool within one GFS filesystem
  - Brocade Silkworm II FC switch

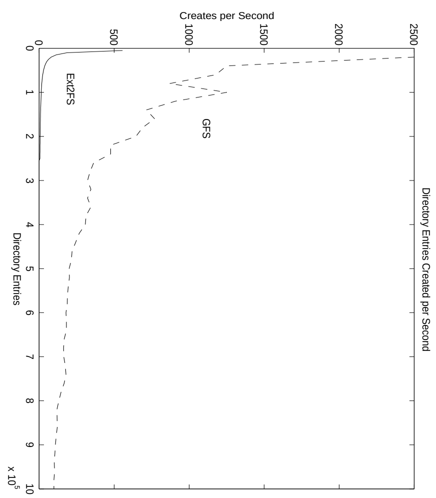
Scalability

- One to four machines are added to a GFS filesystem of constant size
- Workload: 1 million random operations consisting of 50% reads, 25% appends/creates, 25% unlinks
- Each machine performs its workload in separate directory and subpool



Creates per Second

- Comparison of Extendible Hashing directory structure to Linear directory structure
- GFS and Ext2FS both create a million entry directory
- Measured creates per second at constant intervals as directory was filled
- GFS speed levels off due to uncached hash table and leaf blocks



Single Machine Bandwidth

- One Alpha writing to GFS filesystem composed of eight striped disks
- Variable transfer size and request size
  - transfer sizes: 64 KB to 1 GB
  - request sizes: 64 KB to 4 MB
- Writing and reading
  - writing peaked at 50 MB/sec
  - reading peaked at 40 MB/sec

Future Work

- Journaling and recovery
- Growable File Systems
- Some sort of block devices over IP
- Scalability: 4, 8, 16, 32, ... 2^64
- Application level testing: NFS and web serving clusters
- Ports to other OSs (FreeBSD, Solaris, back to IRIX)