



The GFS2 Filesystem

Steven Whitehouse

UKUUG Files and Backup, 2008

Introduction

- Some highlights:
 - 64bit, symmetric cluster filesystem
 - Cluster aware – uses Red Hat cluster suite for cluster services
 - Quorum
 - Locking
 - Journalled metadata (and optionally, data as well)
 - Recovery from failed nodes
 - Mostly POSIX compliant
 - Supports most of the Linux VFS API
 - We don't yet support dnotify for example

Whats new in GFS2?

- Original code size 1176k (approx)
- Current code size 728k (approx)
 - So its shrunk – easier to understand & maintain
- Designed to remove some of the limitations of GFS1
 - Add journals at any time (without expanding the fs)
 - Fixes the statfs problem
 - New “unlink” means unlinked inodes are accounted for on the node(s) which hold the file open, NOT the node which unlinked the inode.
 - Faster (and further improvements are on the list)
 - Fewer internal metadata copies
 - Optimisations of the locking subsystem
 - Reading cached data is now as fast as ext2/3 (hint: its the same code!)
- Its in the 2.6.19+ Linux kernels

Whats new in GFS2? (2)

As a result of upstream review and/or other feedback:

- Different journaled file layout wrt GFS1 (now the same as “normal” files on-disk)
 - Allows mmap(), splice() etc to journaled files
- A metadata filesystem rather than ioctl(s)
- Now all locking is at the page cache level (for GFS1 it was at the syscall level)
 - Faster, supporting new syscalls, e.g. Splice() is much easier
- readpages() support, some writepages() support
 - To be expanded in future
 - Also bmap maps large blocks (no allocs) – again to be expanded
- Supports the “ext3 standard” ioctl via lsattr, chatter

The history of GFS & GFS2

- Originally GFS was a research project at the University of Minnesota
- It came out of work on ocean current simulation
 - Needed to store large amounts of data
 - Needed to allow for simultaneous access from a number of nodes
- Original version of GFS was on SGI's IRIX and used SCSI reservations
- In late 1999/early 2000 Sistina Software took over development of GFS
- Sistina was acquired by Red Hat in late 2003
- I had worked on some of the parts of the original GFS and joined Red Hat in late 2005
- I took over the work on GFS2 from Ken Preslan, who was the original author of most of the code in GFS.

The on-disk format

- A gap (historical)
- The superblock
 - Pointers to locations of “root” and the root of the metafs (gfs2)
- Resource groups (think ext2/3 block groups):
 - A resource group header
 - Bitmaps (two bits per block)
 - Allocated/free
 - Data/metadata (inode)
 - “Free metadata” state originally unused in gfs2, now used for “unlinked, but still open” inodes
 - The data/metadata blocks

Inodes

- Very similar to those of GFS
 - Retains common fields at the same offsets
 - One inode per block
 - Spare space used for either data (“stuffed” inodes) or indirect pointers as required.
 - Indirect pointers formatted as an equal height metadata tree (constant time access to any area of the file independent of offset)
 - Due to the metadata header in each indirect block, the number of pointers per block isn’t a power of two :(
 - Attributes are now set/read by the `lsattr` and `chattr` commands.
 - Extended attributes use either one extra block, or a single layer metadata tree similar to the main metadata tree.

The Directory Structure

The original paper upon which the GFS2 directory structure is based is:
"Extendible Hashing" by Fagin, et al in ACM Trans. on Database Systems,
Sept 1979.

- Small directories are packed into the directory inode
- Hashed dirs based on extendible hashing (fast lookup), but
 - There is a maximum hash table size
 - Beyond that size, the directory leaf chains grow
- We use GFS2's own hash (a CRC32) in the dcache to reduce the number of times we hash a filename (unlike GFS1)
- Each directory leaf block contains an unordered set of "dirents" very much like ext2/ext3

The metadata filesystem

- In GFS they are “hidden” and accessed via an ioctl
- In GFS2 they are accessed by a filesystem of type gfs2meta
- The rindex holds the locations of rgps
- The jindex holds the locations of the journals
 - A file in GFS1
 - A directory in GFS2
- The “per_node” subdirectory holds a set of files for each node in GFS2
 - Inum – Inode number generation
 - Statfs – For fast & fuzzy statfs
- Quota files
- The GFS2 system is extensible so that further files can be added as required.

Locking

- Three lock modes:
 - Shared, Exclusive, Deferred (shared but not compatible with Shared)
 - Deferred is used for Direct I/O to ensure its cluster coherent
 - The DLM's NULL lock mode is used to maintain a ref count on LVBs (currently only used for quota data)
- Pluggable locking architecture
 - lock_nolock
 - A “no-op” lock manager used with local filesystems
 - lock_dlm
 - A full cluster aware lock manager
 - Works with the kernel's DLM and Red Hat clustering software
 - Provides VAX-like DLM semantics translated into GFS's requirements

NFS

- The NFS interface has been designed with failover in mind
 - Requires use of “fsid” export option since the same device might have different device numbers on different nodes
 - The file handles are all byte swapped to be endian independent
- The fcntl locks used by NFS are handled by a user space daemon: gfs2_controld
 - Each node maintains a view of all the fcntl locks in the cluster
 - Uses openais to synchronise locks between nodes
 - Code to allow cluster lock failover is already upstream

Application Writers' Notes

- A quick guide of what to do and what to avoid to get the best performance from a GFS2 filesystem
- Two basic rules:
 - Make maximum use of caching (i.e. don't bounce caches around the nodes unless you really have to)
 - Watch out for lock contention (see above!)
- Example: A distributed email server
 - Lots of small files being created and deleted
 - Need to ensure locality to a node for a particular user
 - Solution:
 - Give each user their own directory
 - Group users to a "home" node in the normal case (using DNS, virtual IP, or similar)

Application Writers' Notes - 2

- `fcntl` caveat
 - When using `F_GETLK` a PID will be returned, but it might exist on any node in the cluster! Be careful
 - Currently there is no way to work out which node the process is on which holds blocking locks
 - We do not support leases
 - We do not support `dnotify/inotify` (except on the local node)
- It is also possible to ask for “local” `fcntl` locking even when running as a cluster filesystem.
- `flock` is fully supported across the cluster. Use this in preference to `fcntl` if your application supports it.
- Using the DLM from an application
 - The DLM is available for application use and this is probably a better solution than `fcntl` locking.

Future Developments

- `readdir`
 - Various speed ups planned
- `dnotify`, leases – if we can work out an efficient way to support it cluster wide
- Performance:
 - Investigations into various areas of performance
 - Currently looking into file creation via `open`
- Stability
 - Continuously adding more tests to the test suite
 - Aiming at ease of maintenance
 - Reducing code size
 - Trying to use Linux idioms wherever possible
 - Reducing interdependency of subsystems

How to get GFS2

- Currently the most up to date distribution is Fedora
 - cman – Contains the cluster infrastructure
 - gfs2-utils – Contains filesystem utilities
- GFS2 is part of the standard Fedora kernel
- Need to write an `/etc/cluster.conf` for clustered use (see the man page)
- Ubuntu also ship GFS2
- Two upstream git trees:
 - -nmw “next merge window” contains everything
 - -fixes used occasionally to pull out bug fixes to send to Linus from time to time.
- Tools are in the cluster CVS