An Overview of The Global File System

Ken Preslan   Sistina Software

kpreslan@sistina.com

David Teigland   University of Minnesota

teigland@borg.umn.edu

Matthew O'Keefe   University of Minnesota

okeefe@borg.umn.edu

http://www.globalfilesystem.org

Outline

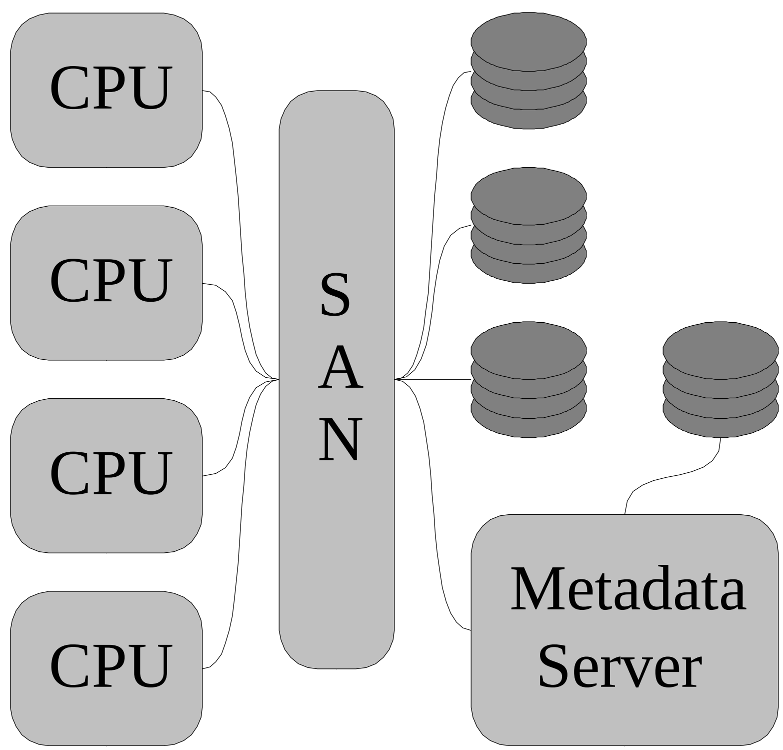
- Network Attached Storage, Fibre Channel, and Shared Disk File Systems
- The Global File System
    - The Network Storage Pool
    - The File System
- Journaling and Distributed Recovery
- Performance
- Future Work

Shared Disk File Systems (SDFS)

- Realize the potential of SANs by coordinating shared access to storage devices
- Each machine accesses the disks as if they were local
  - Faster access
  - Greater availability
- Need a method of synchronization
  - 3^{rd} Party Transfer (Asymmetric)
  - Dlocks/GFS (Symmetric)

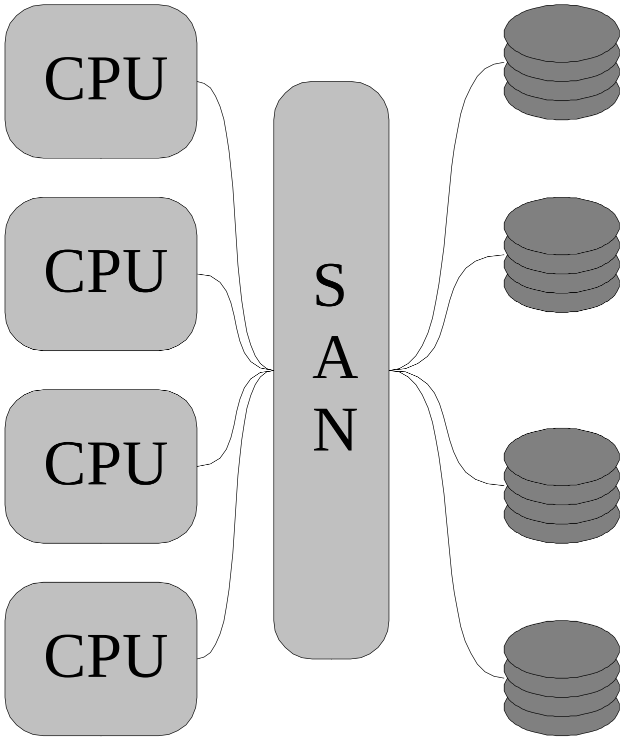
Asymmetric

- 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



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 three parts

  1) The Network Storage Pool Driver

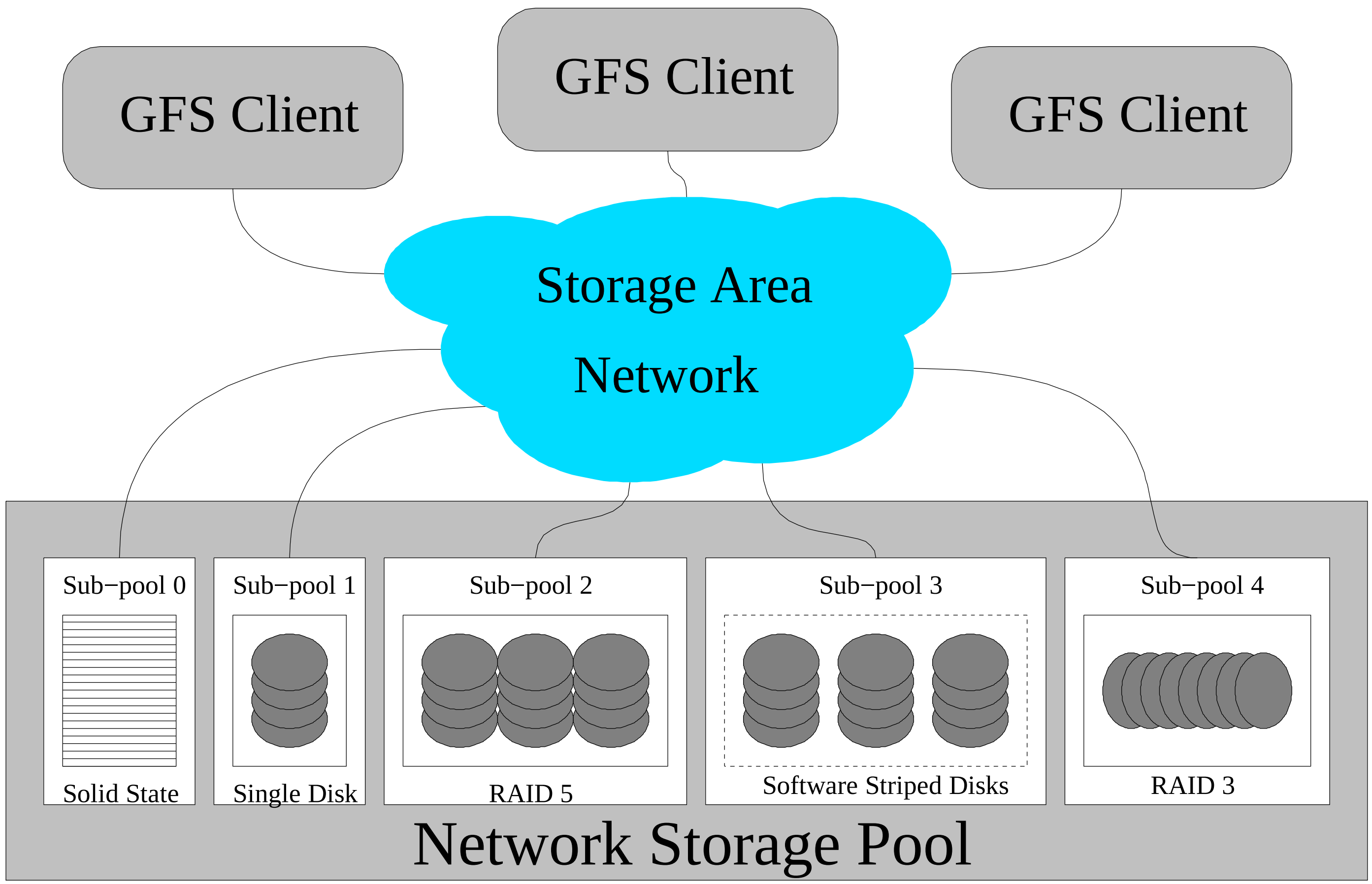
  2) The File System

  3) Locking Modules

The Pool Driver

- A Logical Volume Driver for Network Attached Storage
  - Combines multiple disks into one logical address space
  - Combines multiple Dlock 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



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

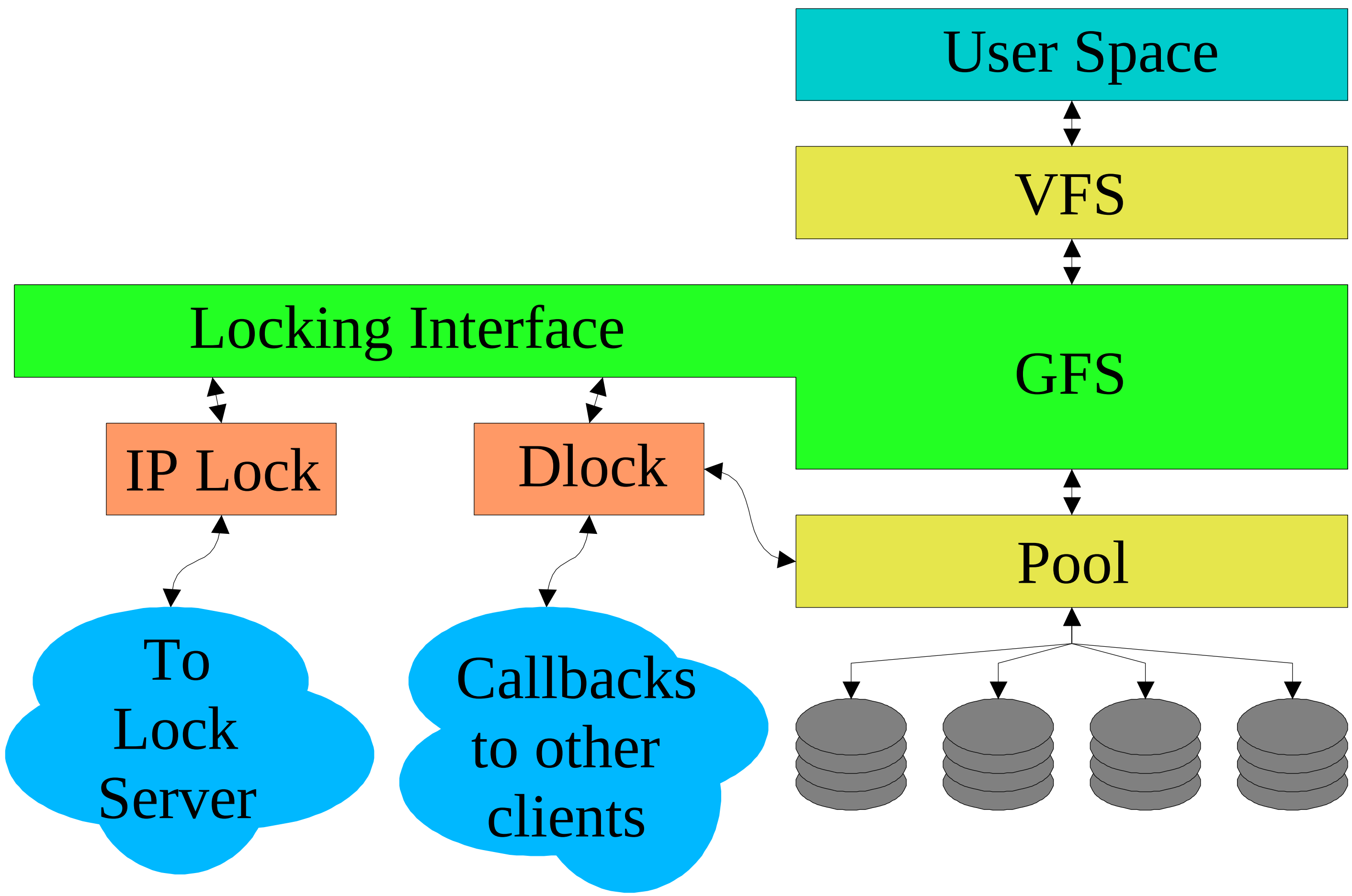
GFS Features

- 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
- Journaled Filesystem

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

Organizational Structure



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

Currently Implemented Protocols

- Nolock − Dummy locks for local file systems
- SCSI Dlock − Locking using SCSI Dlock devices
- GLM − Non−redundant lock protocol over TCP/IP (drives do not need to support Dlock)
- Future: DLM ?

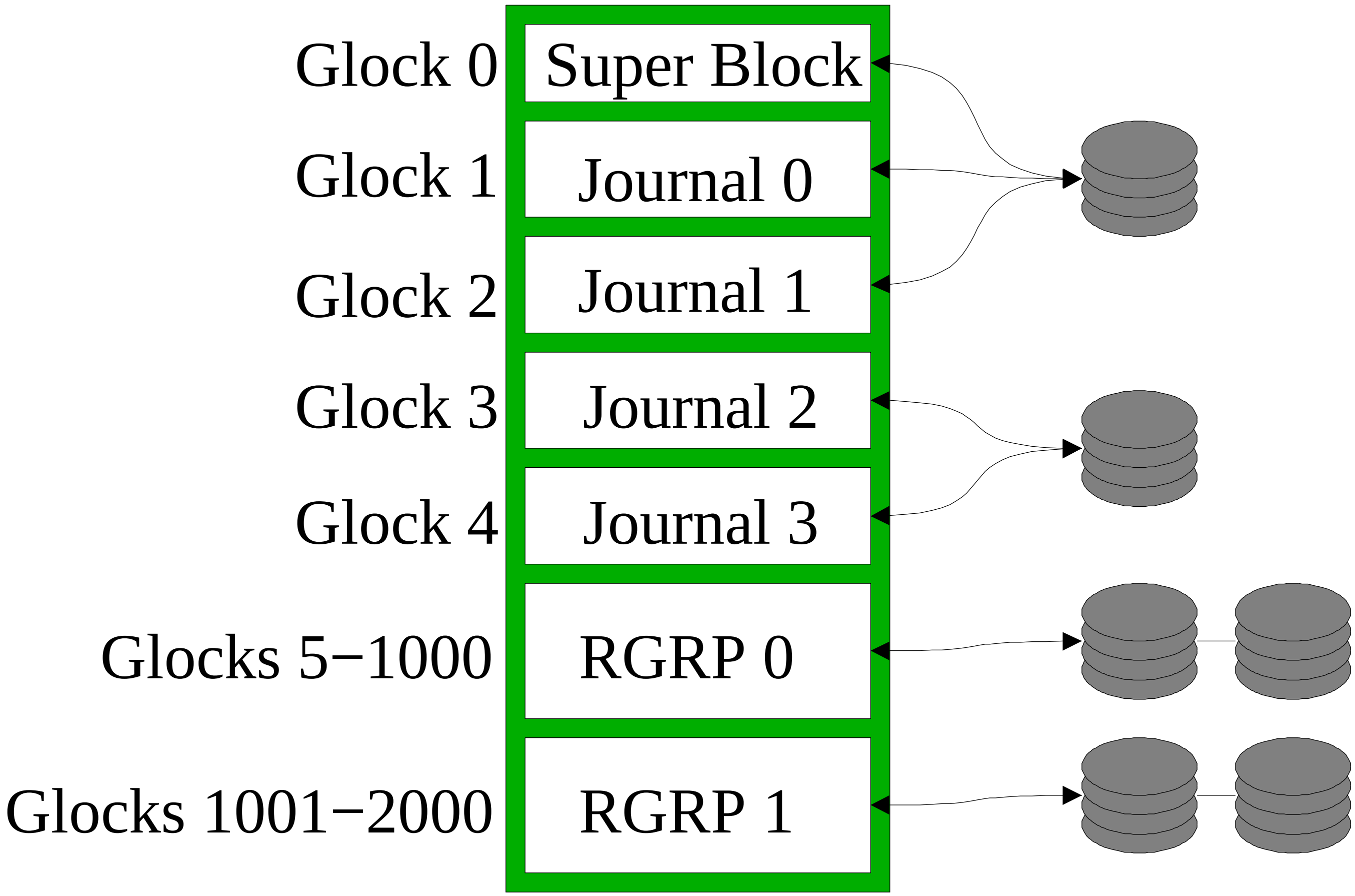
Recovery

- 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

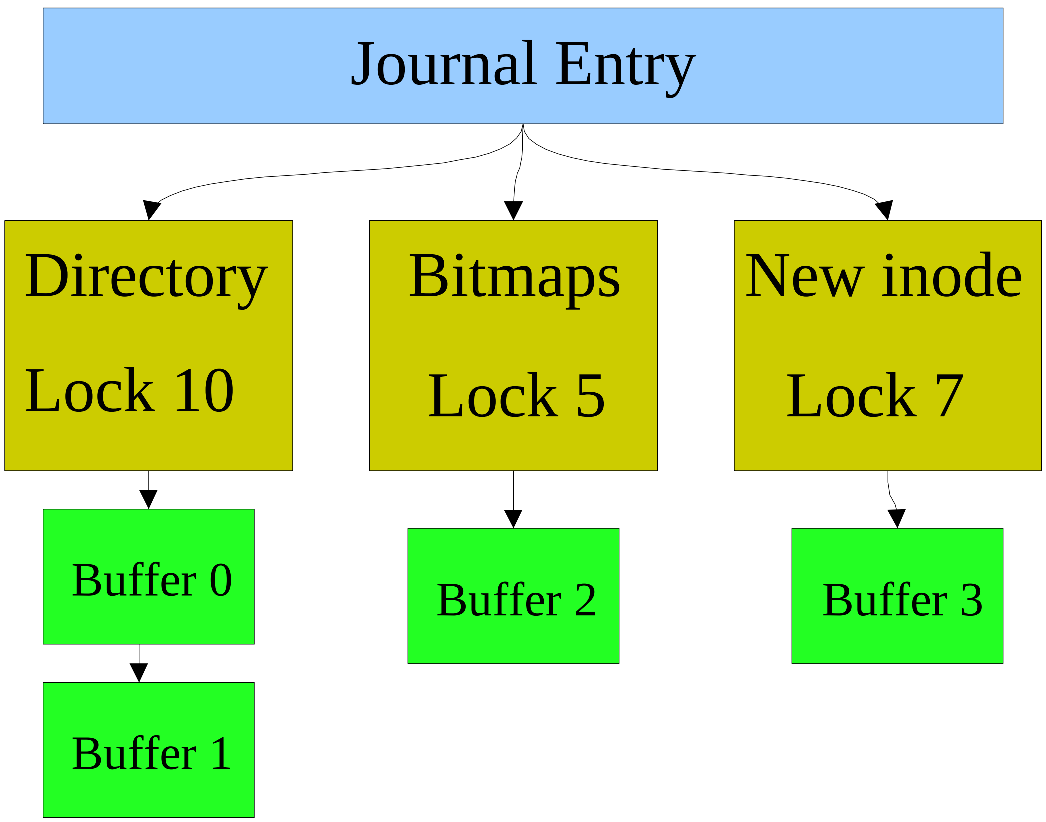


Glock 0
Super Block
Glock 1
Journal 0
Glock 2
Journal 1
Glock 3
Journal 2
Glock 4
Journal 3
Glocks 5−1000
RGRP 0
Glocks 1001−2000
RGRP 1

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
  - 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

Handling Lock Callbacks

- All journal entries are linked to one or more Glocks
- Before Glock is released to another 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)

Handling Lock Callbacks

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

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, the expired client’s ID is passed to a recovery kernel thread
- Machine attempts to begin recovery by trying to acquire journal lock of failed client

Recovery – I/O Fencing

- 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 the journal as recovered
- Mark all expired locks *not expired* for failed client

Replaying Entries

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

Recovery

- The generation number allow journals to be replayed independently
- Allows easy handling of multiple simultaneous machine failures – just recover each journal sequentially
- Machines can continue to work during recovery unless they need a lock which was held by a failed client

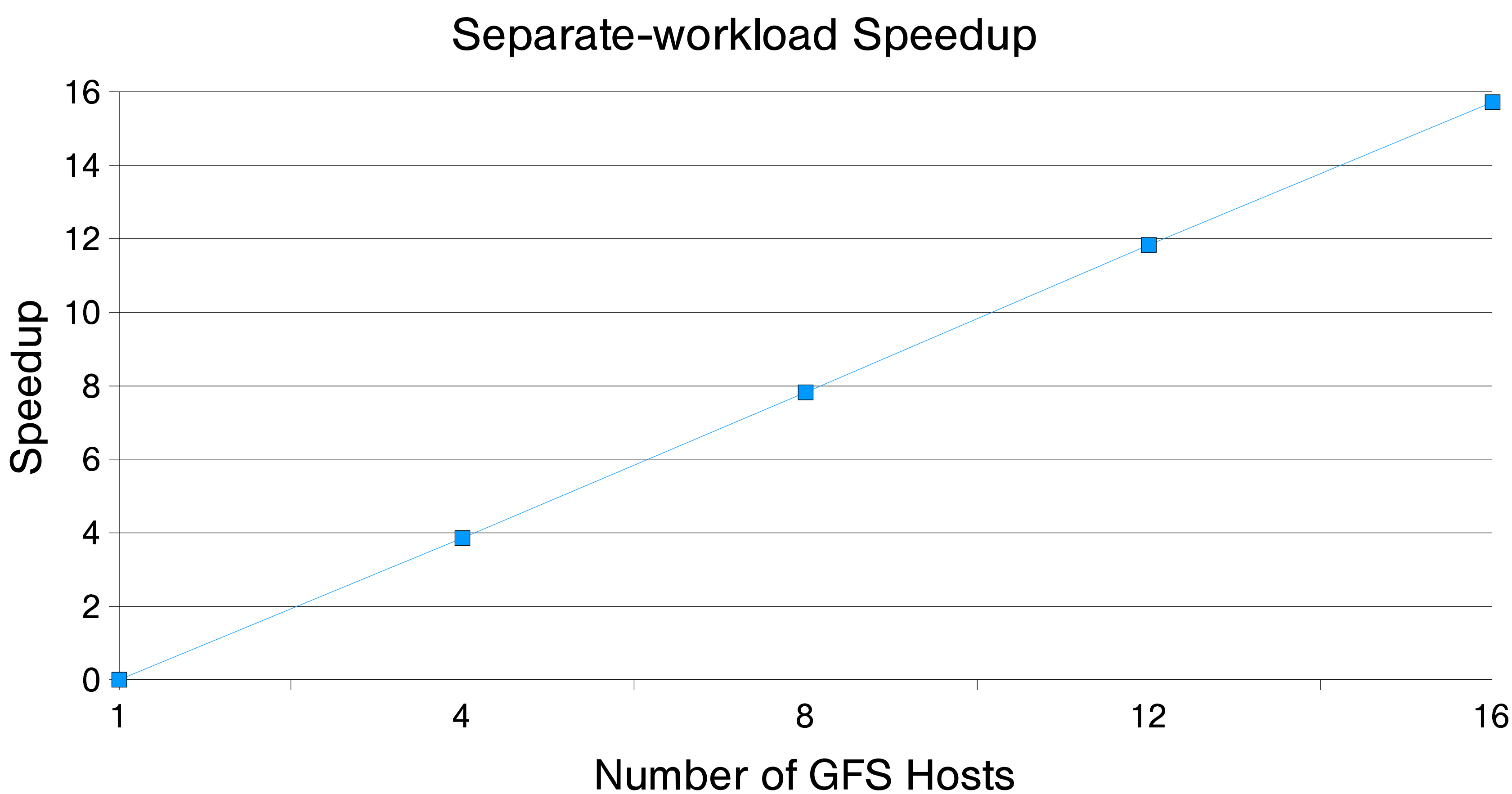
Performance

- Test configuration
  - GFS Antimatter Anteater (non−journaled)
  - 16−node VA Linux cluster
    - PIII, 550 Mhz, 512 MB memory
    - Qlogic 2200 FC adapters
    - 8 eight−disk JBODS (64 drives)
  - Seagate ST39102FC "Cheetah" 9 GB disks
    - Dlock version 0.9.4
    - Each JBOD is a separate striped subpool within one GFS filesystem
  - 4 Brocade 2800 FC switches

Scalability

- One to Sixteen 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

Scaling

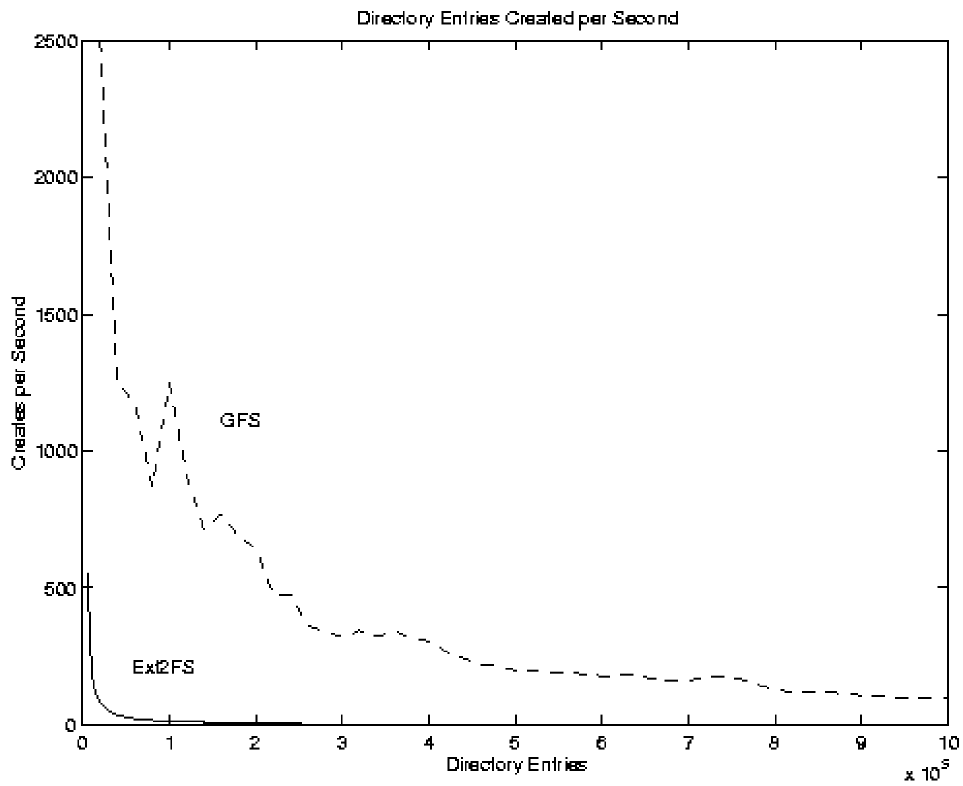


Separate-workload Speedup
Speedup
16
14
12
10
8
6
4
2
0
1
4
8
12
16
Number of GFS Hosts

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

Creates Per Second



Current State / Future Work

- Basic journaling and distributed recovery working
- Speed and reliability improvements soon
- Lots of testing
- Growable File Systems
- Snapshotting
- Scalability: 32, 64, ... 2^64
- Application level testing: NFS and web serving clusters
- Ports to other OSs (FreeBSD, Solaris, back to IRIX)