# Experiences with FUSE in the Real World

Raghavendra Gowdappa, Csaba Henk, Manoj Pillai

February 2019

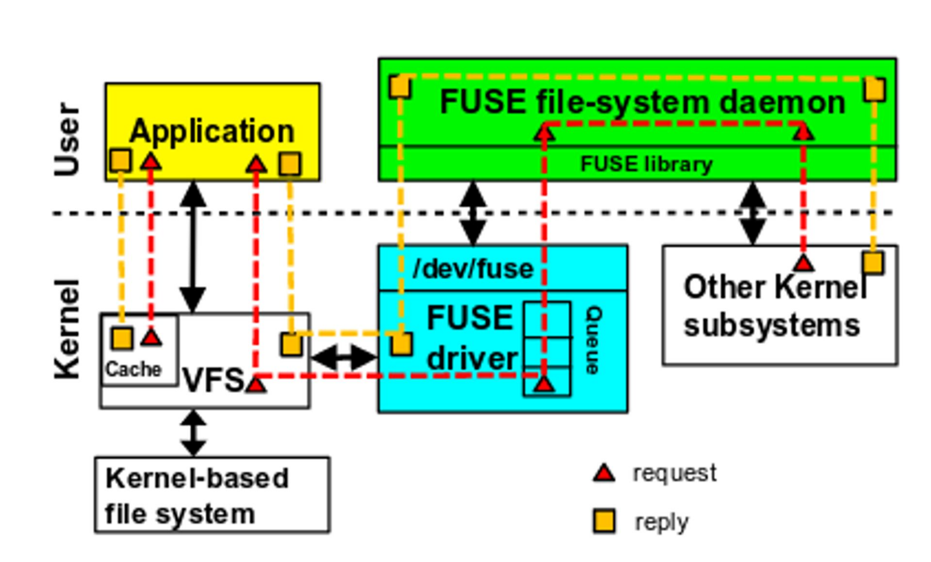
# AGENDA

Implications of FUSE as interface

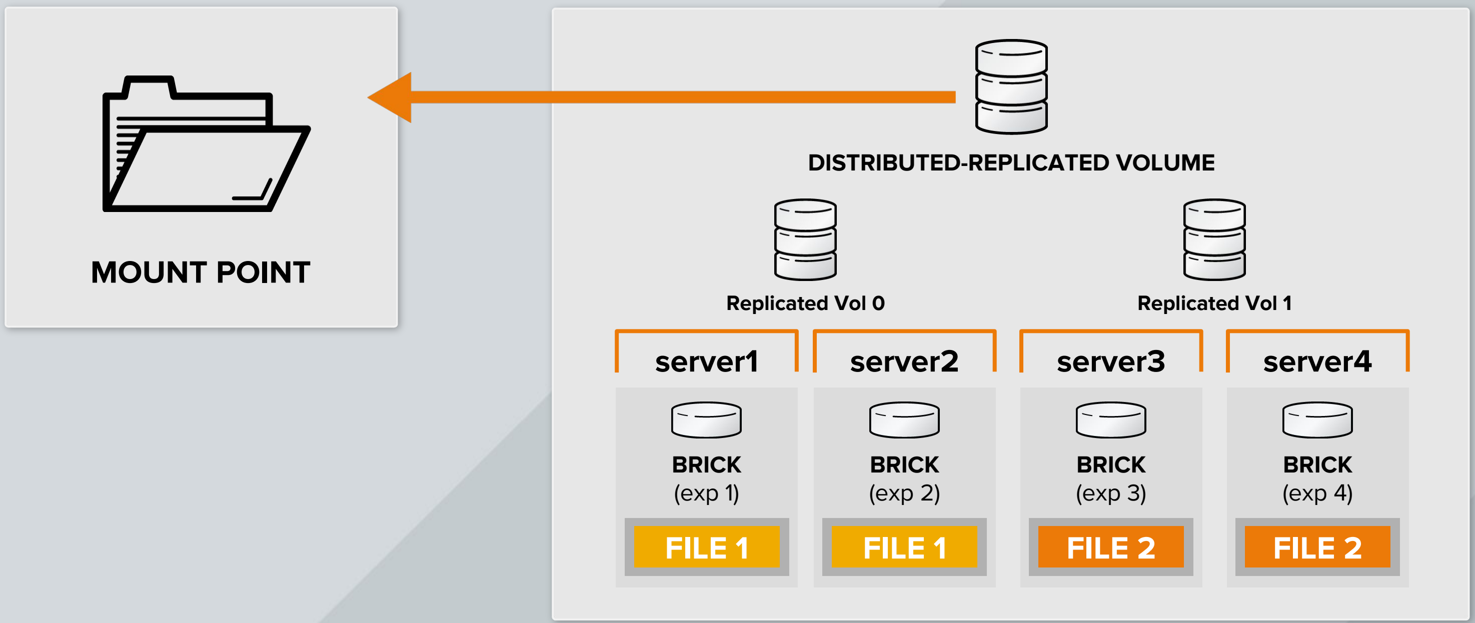
- Impact of latency of FUSE-userspace transitions
- Caching
  - Kernel vs User-Space
  - Write caching and invalidations
  - Directory Entries
- Memory Management
- FUSE areas of improvement

# FUSE architecture

- Perception is that  multiple context switches limits performance
- File-system daemon could be a distributed file system, as in the case of gluster, with network latencies incurred for most operations

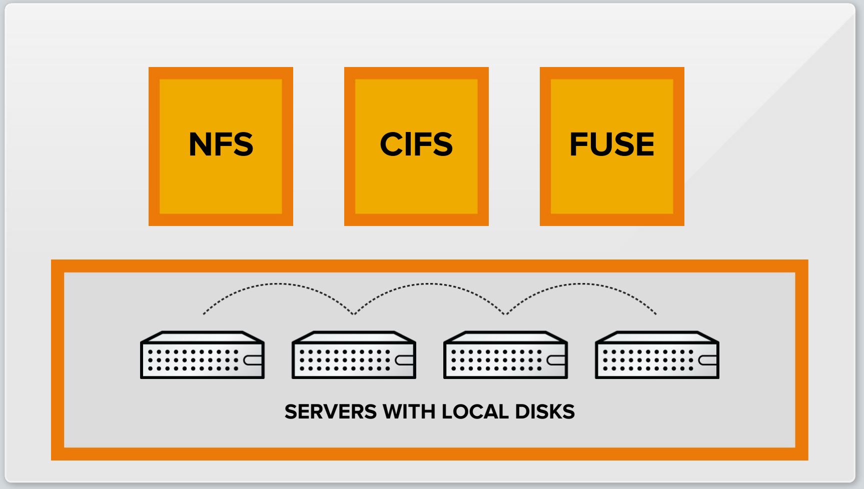


# GLUSTER ARCHITECTURE OVERVIEW



# GLUSTER ARCHITECTURE

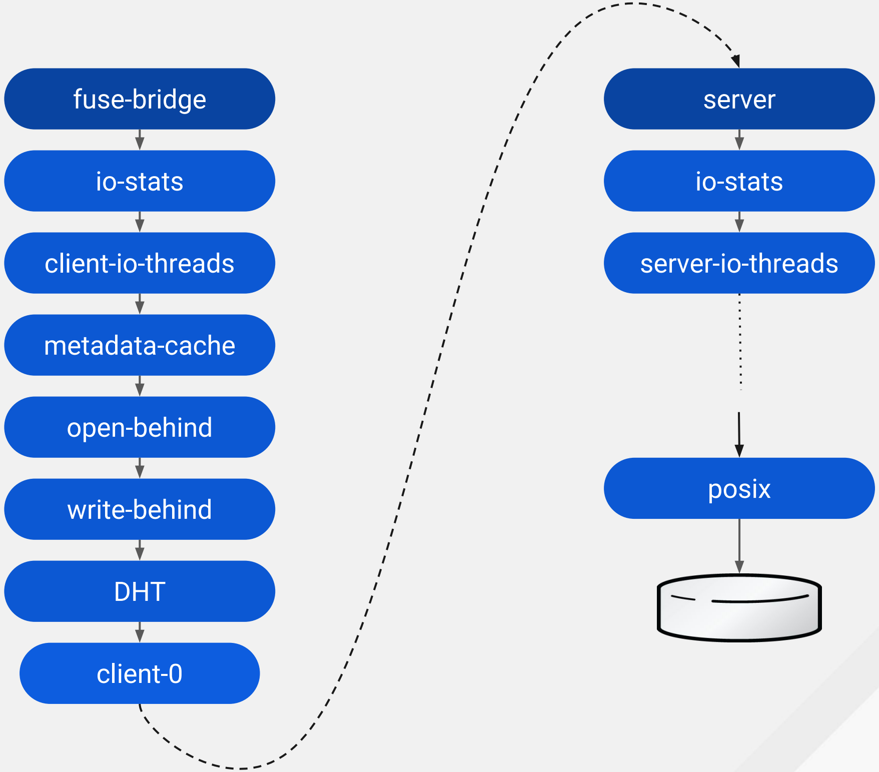
Distributed scale out storage using industry standard hardware



**Aggregates systems to one cohesive unit and presents using common protocols**

# GLUSTER ARCHITECTURE OVERVIEW

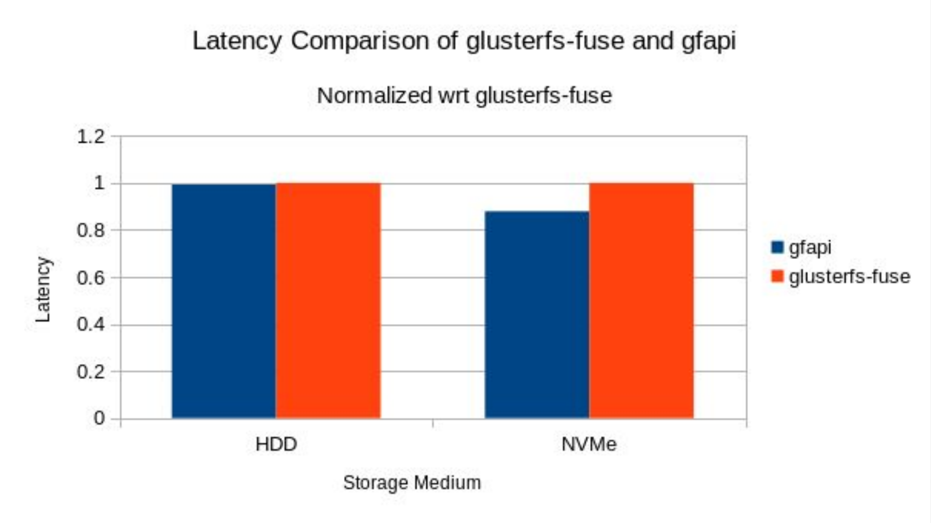
Gluster client-side implements distribution, replication and caching functionality.



# FUSE Latency

Has not been the root-cause of any performance issues experienced with glusterfs-fuse.

Partly because network and storage device latency dominate: graph shows latency of FUSE path compared to gfapi.



# Caching: Kernel vs User-Space

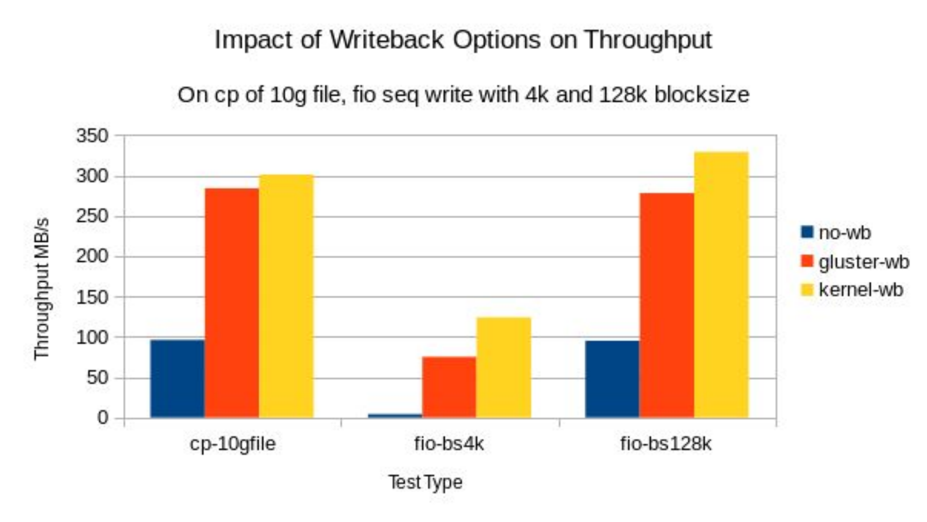
FUSE filesystems can leverage kernel file system caches and functionality.

- Kernel
  - Page cache, read-ahead and writeback
  - Inode and dentry caches
- Gluster implements similar caching functionality and more in user-space
  - Read-ahead, write-behind, io-cache (data cache)
  - Readdir-ahead, currently not available in kernel; could be implemented.
  - Parallel-readdirplus, motivated by specific gluster architectural details

# Kernel vs. User-Space Caches

Which is preferable?

- Kernel caches closer to application; perform better.
- Graph shows kernel fuse-writeback more effective at improving performance, compared to gluster write-behind:
  - 65% better for fio with 4k write size
  - 18% better for fio with 128k write size



For glusterfs-fuse, no compelling case for gluster user-space implementation of read-ahead and data cache (io-cache) over kernel provided equivalent.

# FUSE Write Caching and Invalidation

Relevant FUSE features:

- write-through and write-back caching
- reverse invalidation framework for user-space process to invalidate kernel cache
- flag to control cache retention on open
- auto-invalidation logic for invalidating cache, and auto-invalidation on/off knob
    - only for write-through caching; write-back leaves invalidation responsibility to user-space.
    - stat update results in a check and invalidation of cached pages, if stat times have changed

# glusterfs-fuse: Invalidation and Write Caching

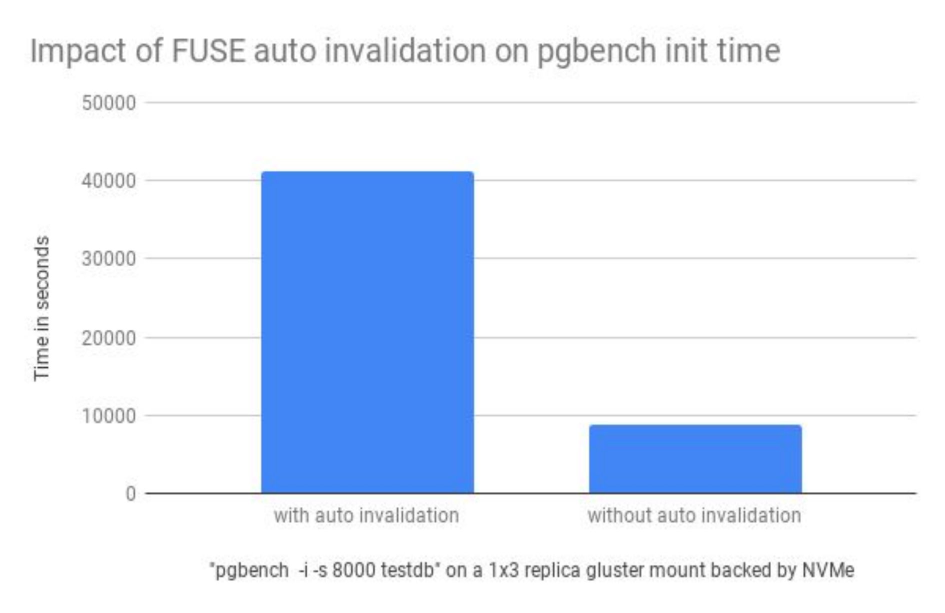
glusterfs-fuse implementation:

- write-through caching is default
- retain cache on open
- fuse auto-invalidation enabled by default
- kernel writeback caching can be enabled
    - mount option: *kernel-writeback-cache=yes*

# Caching and Invalidation

Retaining caches after write in write-through caching

- FUSE auto invalidation not intelligent enough to distinguish local writes from remote writes
- Recommended to rely on custom invalidation policy implementation using reverse invalidation framework



# Custom Invalidation Policies in Userspace File Systems

- Userspace filesystems are recommended to implement custom invalidation policies using reverse invalidation framework for both write-through and write-back modes
  - NFS uses close-to-open consistency
    - Invalidate during open if file stat has changed since last close on the inode
  - Suggestions to use leases for stronger consistency

# Metadata Caching and Prefetching

Certain gluster user-space functionality is needed for good performance

- Functionality needed for good performance given gluster’s distributed nature and metadata-server-less architecture (missing in kernel)
  - Readdir-ahead
  - parallel readdirplus
- FUSE stat invalidation
  - md-cache (stat cache)

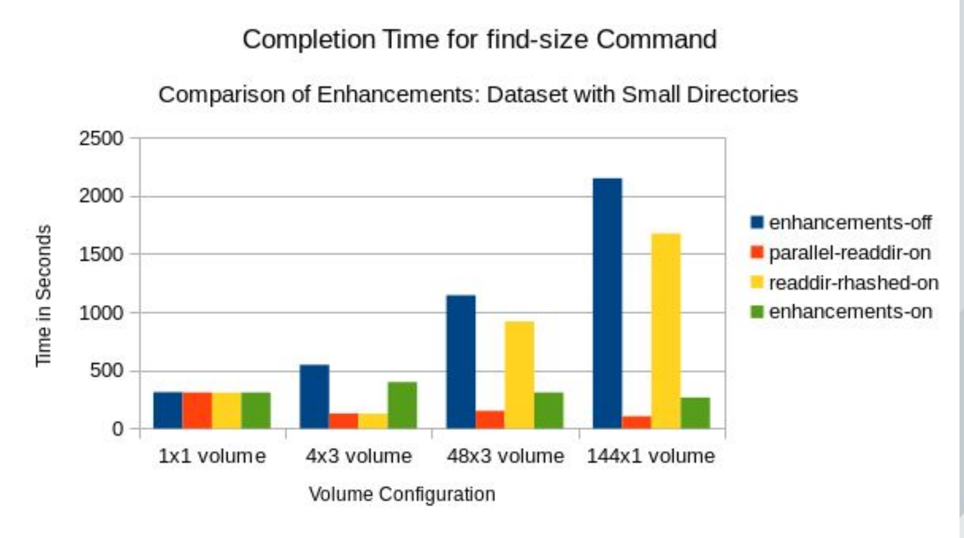
# Impact of parallel-readdirplus on Performance (Vault’17)

**Explanation**

- Dataset with small directories, which was not showing improvement with md-cache on
- Chart shows impact of parameters parallel-readdir and readdir-hashed
  - `performance.parallel-readdir on`
  - `cluster.readdir-hashed on`
- md-cache is on in all cases

**Observations**

- readdir-hashed improves performance on its own, but parallel-readdir=on is better than both set to on



# Memory Management

- Glusterfs has to remember all inodes kernel had looked up and not forgot
  - Inode count can run from 10s of thousands to millions
- Many xlators in glusterfs maintain per xlator state in inodes and fds
  - Caches too are stored as xlator state in inode
  - Coupled with high inode count, memory consumed by inodes can be huge
- Inodes, dentries in userspace filesystem get accounted as process memory
  - OOM kill instead of inode cache eviction
  - Userspace caches can result in OOM kill
  - Glusterfs implemented its own lru based inode garbage collection using FUSE reverse invalidation framework
- Userspace process has limited visibility of global memory consumption
  - Cache memory management is less dynamic

# FUSE areas of improvements

- Max value of read-ahead tunable is 128KB
- Responses of operations that update file can carry latest stat to be populated in attributed cache

# ACKNOWLEDGMENTS

Contributions to this work

- Shekhar Berry and Xavi Hernandez for assistance on root causing pgbench performance to sub-optimal cache functioning on client
- Miklos Szeredi for all the assistance provided on FUSE

# Questions?

Presenters can also be reached at

- rgowdapp@redhat.com
- chenk@redhat.com
- mpillai@redhat.com

# THANK YOU