Optimizing SDS for the Age of Flash

Krutika Dhananjay, Raghavendra Gowdappa, Manoj Pillai @Red Hat

GLUSTER

Agenda

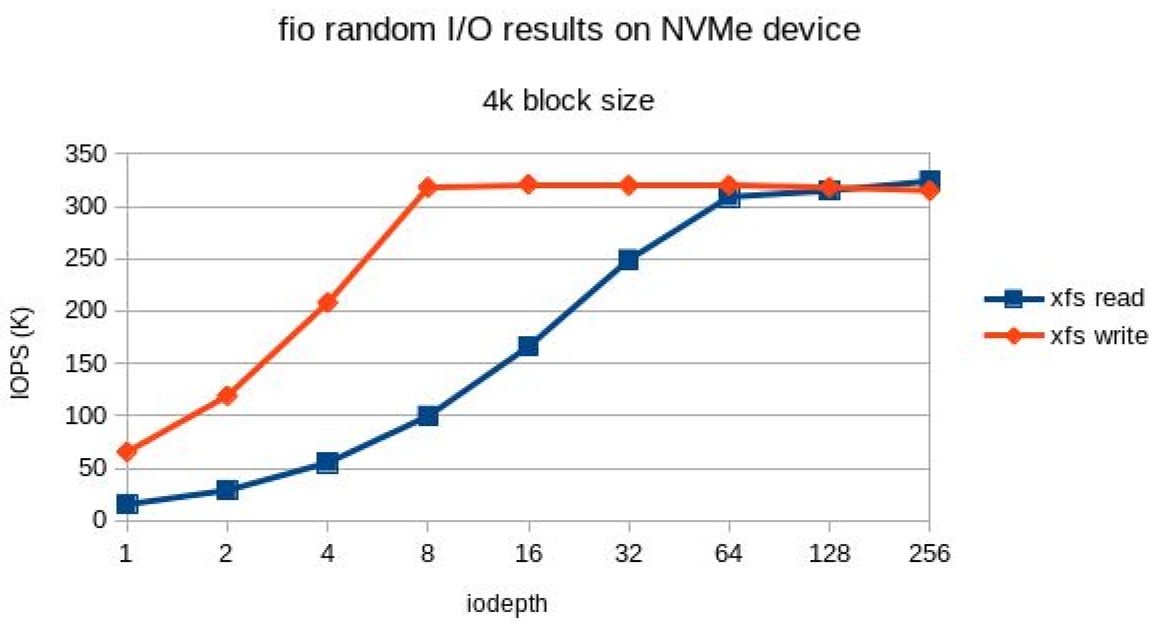
- Introduction and Problem Statement
- Gluster overview
- Description of Enhancements
- Lessons Learned
- Work in Progress

Introduction

- Gluster’s traditional strength: sequential I/O workloads
- New Trends
  - SSDs popularity, particularly for random I/O workloads
    - IOPS capabilities way higher than HDDs
  - Gluster integration with KVM and Kubernetes
    - New workloads including IOPS-centric ones
- Need to ensure that gluster is capable of delivering the IOPS that devices are capable of

Problem Statement

XFS Performance on NVMe



- IOPS increase with iodepth upto device limits
- Able to deliver device capabilities

Random I/O Test

```
[global]
rw=randread
startdelay=0
ioengine=libaio
direct=1
bs=4k
numjobs=4

[randread]
directory=/mnt/glustervol
filename_format=f.$jobnum.$filenum
iodepth=8
nrfiles=4
openfiles=4
filesize=10g
size=40g
io_size=8192m
```

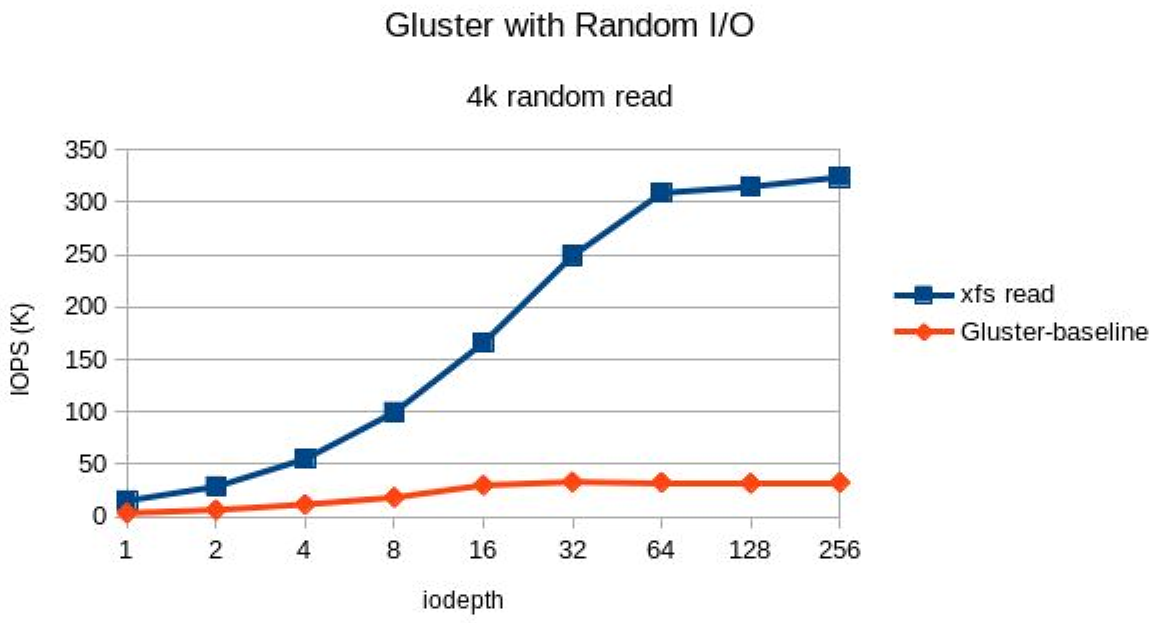
```
[global]
rw=randwrite
end_fsync=1
startdelay=0
ioengine=libaio
direct=1
bs=4k
numjobs=4

[randwrite]
directory=/mnt/glustervol/
filename_format=f.$jobnum.$filenum
iodepth=8
nrfiles=4
openfiles=4
filesize=10g
size=40g
io_size=8192m
```

Configuration

- Systems:
  - Supermicro 1029p, 32 cores, 256GB
  - Single NVMe drive per system
- Software versions
  - glusterfs-3.13.1+enhancements, RHEL-7.4
- Tuning
  - gluster tuned for direct/random I/O
    - strict-o-direct=on, remote-dio=disable
    - stat-prefetch=on
    - most other gluster performance options turned off: read-ahead, io-cache etc.

Gluster Performance on NVMe



- IOPS peak is low compared to device capabilities

What is Gluster?

- Scale-out distributed storage system
- Aggregates storage across servers to provide a unified namespace
- Modular and extensible architecture
- Layered on disk file systems that support extended attributes
- Client-server model

Gluster - Terminology

BRICK

The basic unit of storage

SERVER/NODES

Contain the bricks

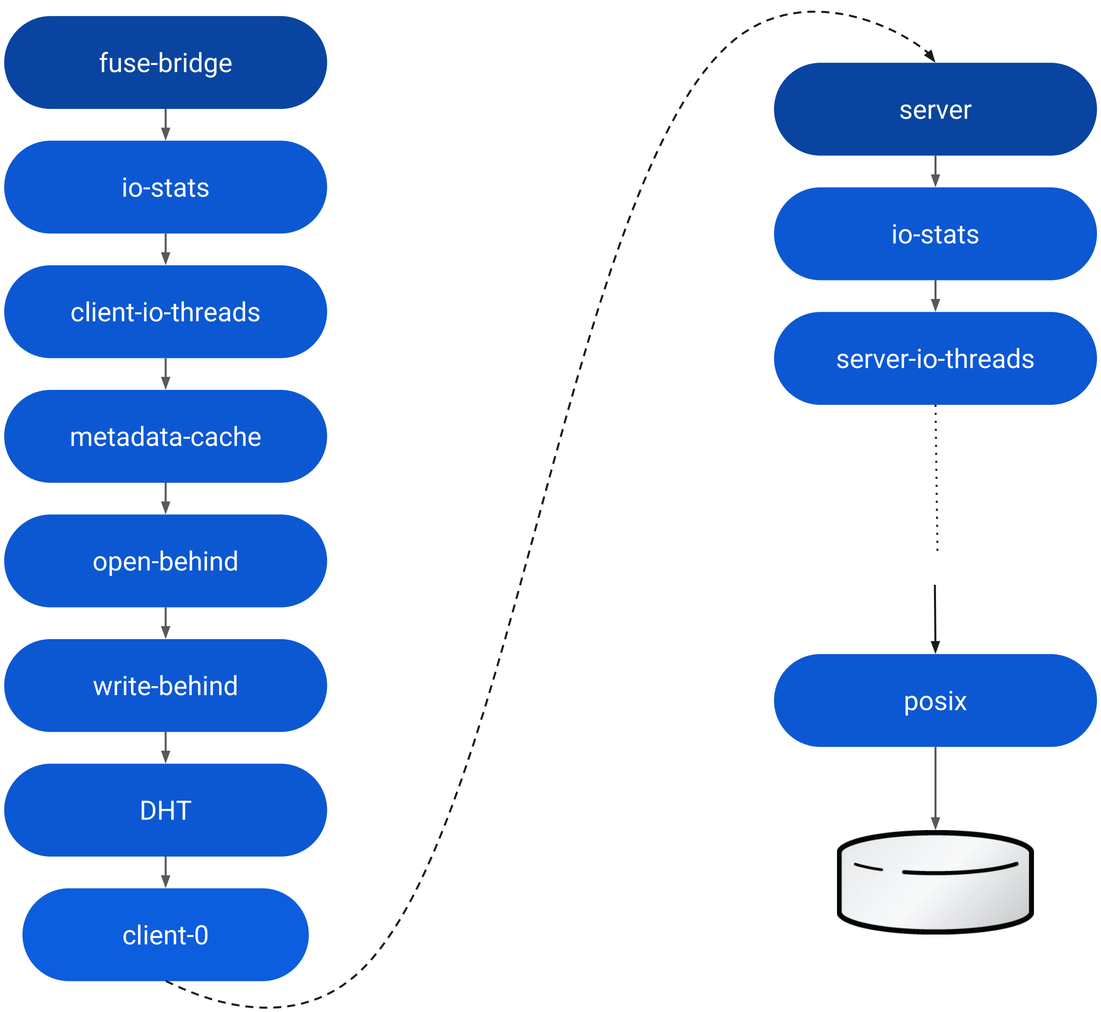
VOLUME

A namespace presented as a POSIX mount point

TRANSLATOR

Stackable module with a specific purpose

Gluster Translator Stack



Gluster threads and their roles

Fuse reader thread

- Serves as a bridge between the fuse kernel module and the glusterfs stack
- “Translates” IO requests from /dev/fuse to Gluster file operations (fops)
- Sits at the top of the gluster translator stack
- Number of threads = 1

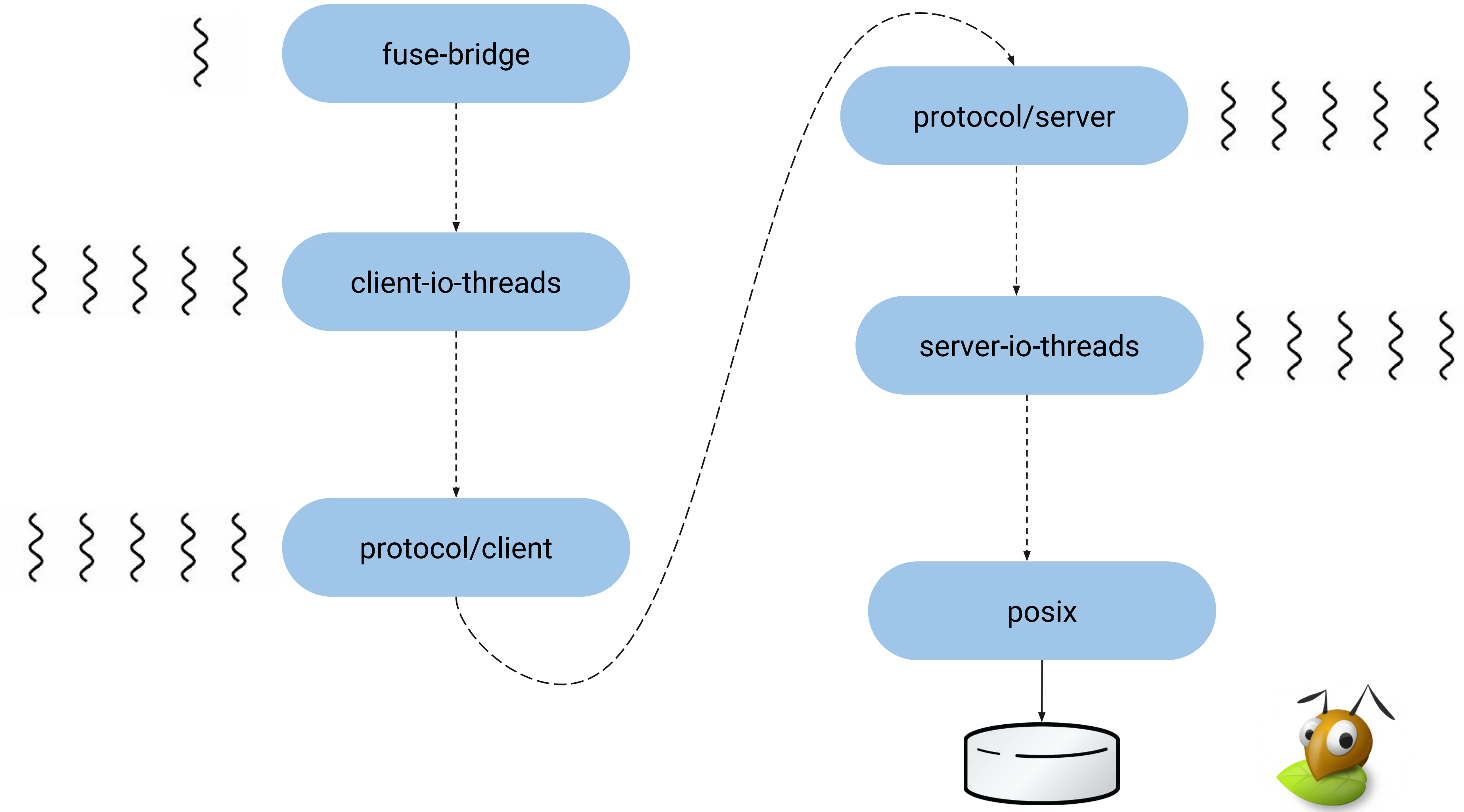
io-threads

- Thread-pool implementation in Gluster
- The threads process file operations sent by the translator above it
- Scales threads automatically based on number of parallel requests
- By default scales up to 16 threads.
- Can be configured to scale up to a maximum of 64 threads.
- Loaded on both client and server stack

Event threads

- Thread-pool implementation in Gluster at socket layer
- Responsible for reading (writing too in some cases) requests from the socket between the client and the server
- Thread count is configurable
- Default count is 2
- Exist on both client and server

Piecing them together...



Too many threads, too few IOPs...

- Enough multi-threading in the stack to saturate spinning disks
- But with NVMe drives, hardware was far from saturated
- Experiments indicated that the bottleneck was on the client-side.
- Multi-threading + global data structures = lock contention

Mutrace to the rescue...

- Mutrace is a mutex profiler used to track down lock contention
- Provides a breakdown of the most contended mutexes
  - how often a mutex was locked
  - how often a lock was already taken when another thread tried to acquire it
  - how long during the entire runtime the mutex was locked

Performance debugging tools in Gluster

- Volume profile command - provides per-brick IO statistics for each file operation.
  - Stats include number of calls, min, max and average latency per fop, etc
  - Stats collection implemented in io-stats translator
  - Can be loaded at multiple places on the stack to get stats between translators.
  - Experiments with io-stats indicated highest latency between client and server translator

Description of Enhancements

Fuse event-history

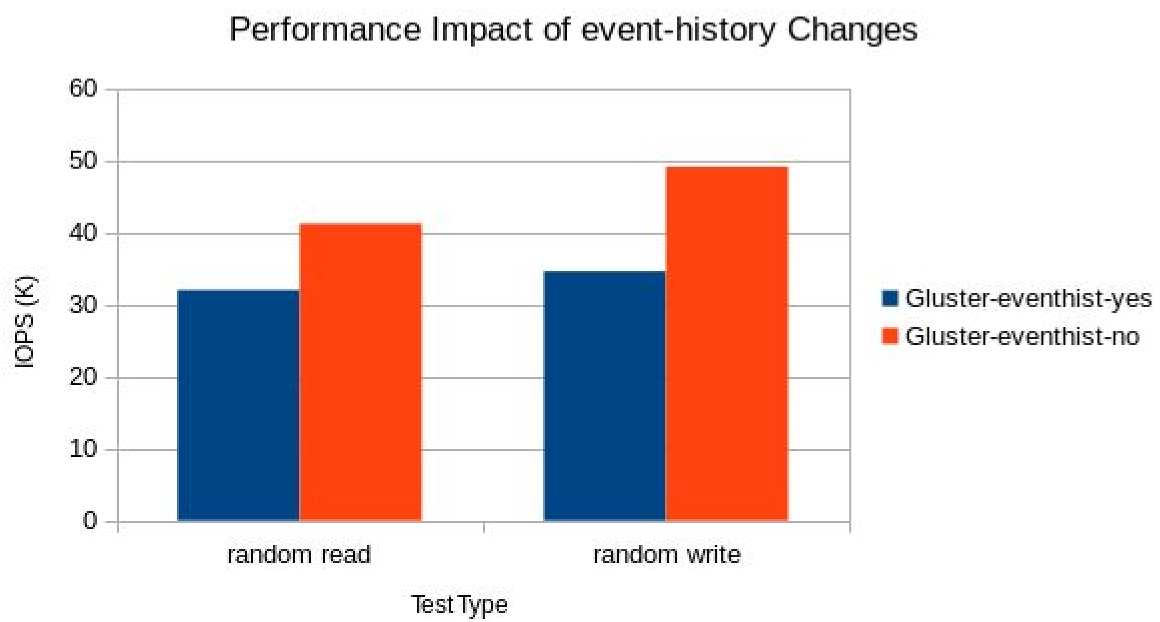
PROBLEM

- Fuse-bridge maintains a history of most recent 1024 operations it has performed in a circular buffer
- Tracks every fop in request as well as response path
- Protected by a single mutex lock
- Caused contention between fuse reader thread and client event thread(s)

FIX

Disabled event-history by default since it is used only to trace fops for debugging issues.

Impact of disabling event-history



- Random read IOPs improved by ~ and random write IOPs by ~15K.

Scaling fuse reader threads

PROBLEM

After removing the previous bottlenecks, fuse reader thread started consuming ~100% of CPU

FIX

Added more reader threads to process requests from /dev/fuse in parallel

IMPACT OF FIX

IOPs went up by 8K with 4 reader threads.

iobuf pool bottleneck

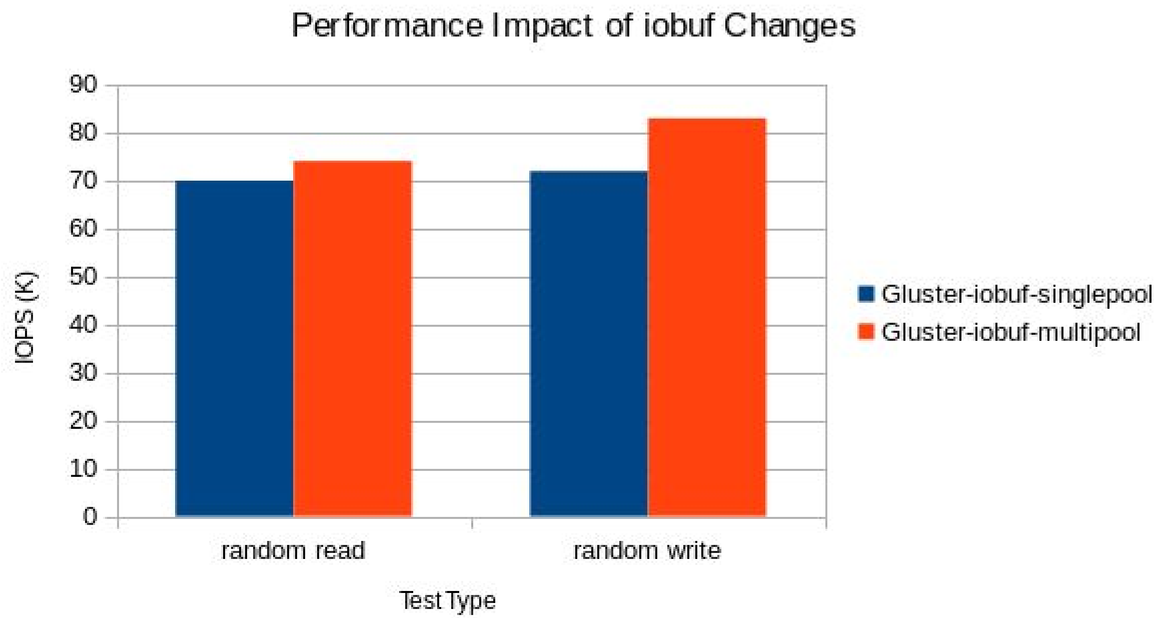
PROBLEM

- Iobuf - data structure used to pass read/write buffer between client and server
- Implemented as a preallocated pool of iobufs to avoid the cost of malloc/free every time
- Single global iobuf pool protected by a mutex lock
- Caused lock contention between fuse reader thread(s) and client event threads

FIX

- Create multiple iobuf pools
- For each iobuf allocation request, select a pool at random or using round-robin policy
- Instead of all threads contending on the same lock, the contention is now distributed across iobuf pools
- More pools implies fewer contentions

Impact of iobuf Enhancements



- Random read IOPs improved by ~4K and random write IOPs by ~10K.

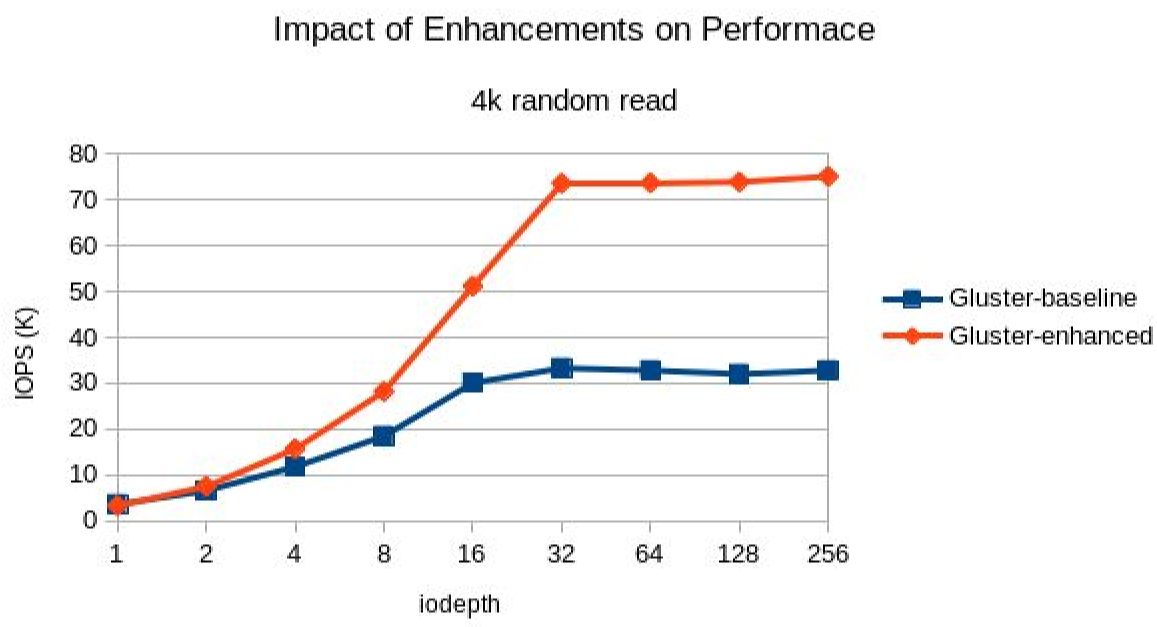
rpc layer

- Multithreaded “one-shot” epoll-based one non-blocking socket connection between a single client and a brick
- Profile information showed high latencies in rpc layer
- Tried increasing concurrency between request submission and reply processing within a single rpc connection
    - No gains
- An earlier fix had shown that reducing the time a socket is not polled for events improves performance significantly
    - Maybe the bottleneck is while reading msgs from socket?

rpc...

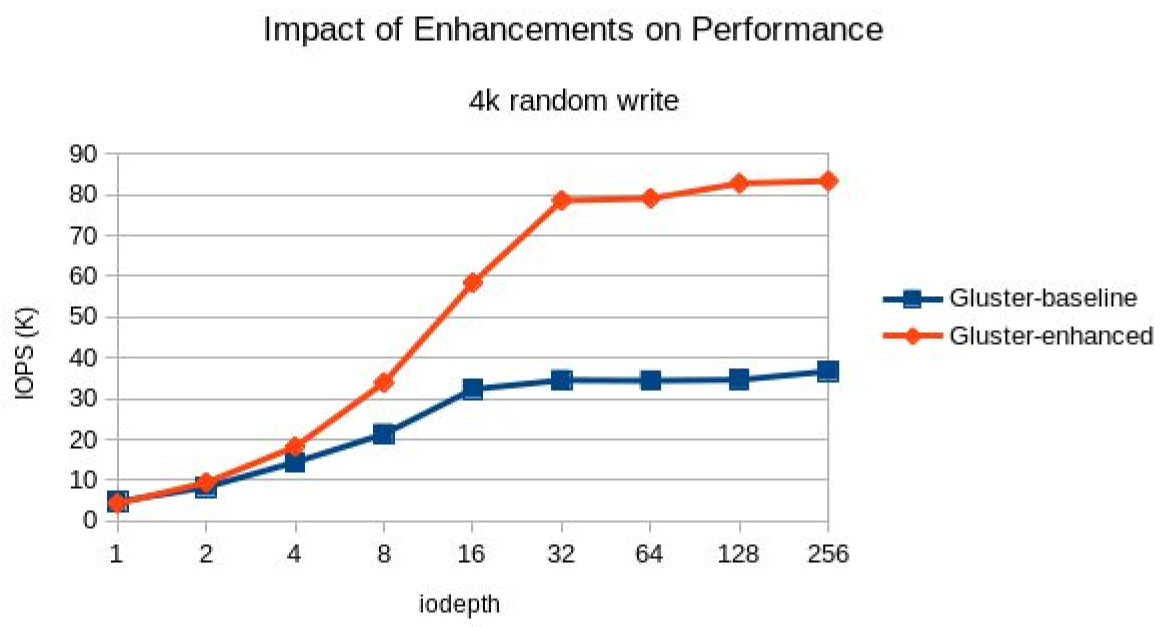
- Scaling to 3-brick distribute showed improvement
  - Is single connection b/w client and brick the bottleneck?
- Multiple connections between a single brick and client gave same improvement as 3-brick distribute
  - Credits - "Milind Changire" <mchangir@redhat.com>

Impact of Enhancements



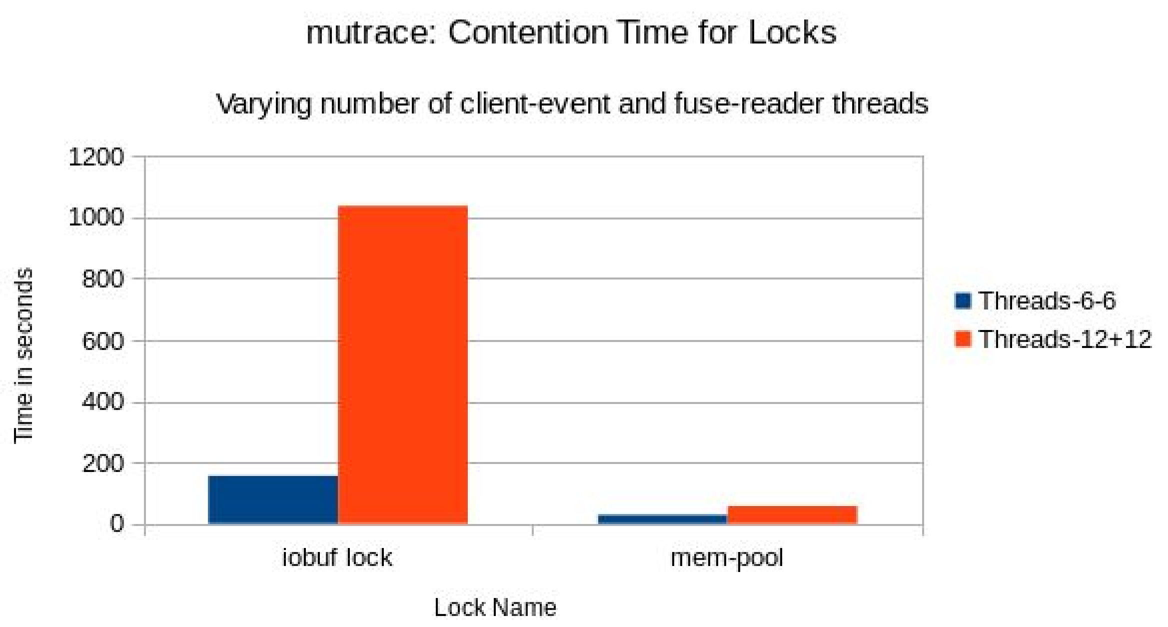
- Random read IOPS peaks around 70k compared to ~30k earlier

Impact of Enhancements



- Random write IOPS peaks at about 80k compared to less than 40k earlier

Lessons learnt



- Highly contended locks, which one affects performance?
  - Hint: multiple datasets collected by altering parallelism

Lessons learnt

- During highly concurrent loads, multiple threads are necessary even for a lightweight task
  - Client-io-threads vs fuse reader threads
- Need more lightweight tools
  - Mutrace slows down tests significantly, potentially skewing information on bottlenecks
- Multiple bottlenecks. Validating fixes require careful analysis
  - Process of analysis has to be iterative

lessons...

- Multiple incremental small gains added up to significant number
- Simple tools like systat utilities like top gave good insights
- Significant time spent in micro-optimization
    - Efforts adding more concurrency between request submission and reply reading in rpc
    - High level models were helpful to (dis)prove hypothesis even before attempting fix

Future Work

- Bottleneck analysis on both client and bricks still a work in progress
  - Work till now concentrated on client
- Spin Locks while reading from /dev/fuse wasting CPU cycles
- Reduce lock contentions
  - Inode table
- Working towards lightweight tracing tools for lock contention

Future...

- Evaluate other rpc libraries like grpc
- Zero copy using splice
  - https://github.com/gluster/glusterfs/issues/372
- Analyse the impact of a request or reply having to pass through multiple thread subsystems
  - Fuse-reader threads vs Io-threads vs event-threads vs rpcsvc-request-handler threads vs syncenv threads
- Get all the work merged into master :)
  - https://bugzilla.redhat.com/show_bug.cgi?id=1467614

Thanks!!