Red Hat Performance Briefs

RHGS 3.1 Performance Brief

Shekhar Berry
Ben England
Manoj Pillai
Ben Turner

Version 1.0

October 2015

redhat.

1801 Varsity Drive™
Raleigh NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Send feedback to refarch-feedback@redhat.com

Table of Contents

1. Executive Summary

RHGS (Red Hat Gluster Storage) 3.1 is another step towards delivering on the vision of scalable, highly durable, high-performance storage envisioned in the Gluster architecture. This document summarizes the current state of RHGS performance, and discusses significant changes in the 3.1 release that have important performance implications for end-users.

2. Performance of Core Features

This section presents performance results for RHGS 3.1 for the following configuration:

- distributed-replicated volume
- bricks on RAID-6 devices, replica-2 configuration
- clients mounting volume using the fuse module (gluster native protocol)

This combination represents a basic configuration for RHGS which has been available and supported since the earliest releases, and is still the most commonly used configuration. This section presents performance results for small-file and large-file workloads, which represent two broad workloads classes that present different challenges to a storage solution. Hence, the results in this section serve as a good summary of the current state of performance for RHGS core features.

2.1 Small-file Performance

This section presents performance results for the RHGS 3.1 release for small-file use cases. The results here demonstrate greatly improved performance compared to older releases. The improvements for small-file workloads mostly come from the following enhancements:

1. An optimization for file lookups, which improves the performance of negative lookups by avoiding a lookup on every subvolume. This optimization is new in RHGS 3.1 and is available as a tunable parameter on a volume. The syntax and recommended tuning is given below.

   gluster volume set <volume name> cluster.lookup-optimize on

   The default value for cluster.lookup-optimize is *off*.

2. The multi-threaded epoll enhancement, introduced in RHGS 3.0.4, which allows multiple event threads to process requests in parallel, eliminating some of the single-thread bottlenecks in earlier RHGS versions. There is a tunable parameter to specify the number of event threads to be used on the server side, and a separate tunable parameter to specify the number of event threads on the client side. The syntax and recommended tuning is given below:

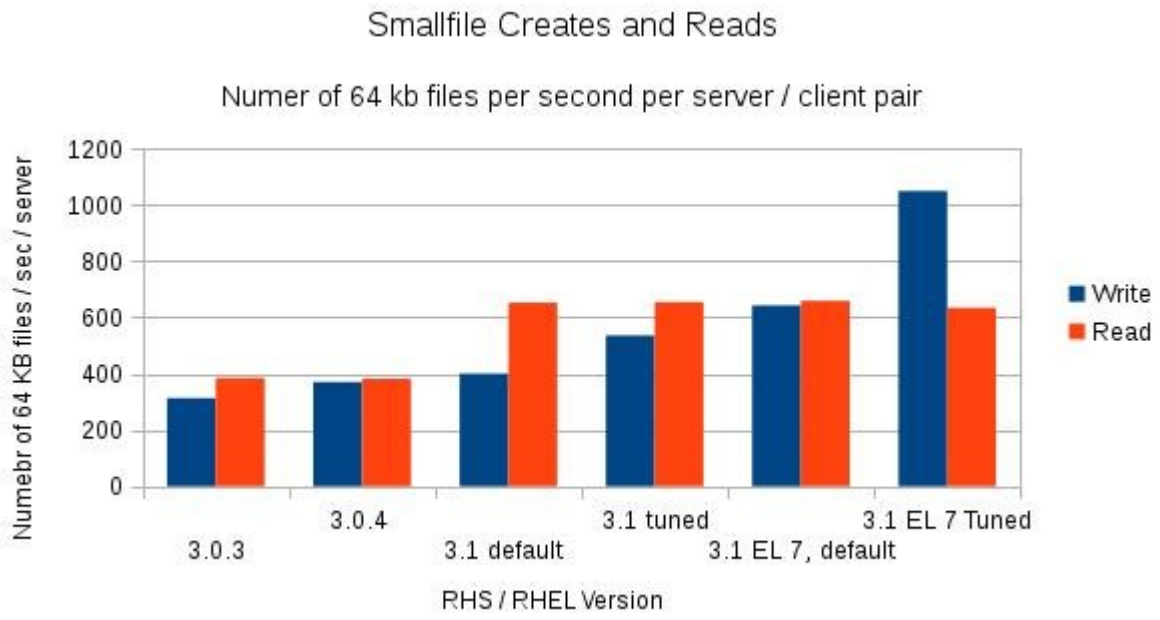
   gluster volume set <volume name> server.event-threads 4

   gluster volume set <volume name> client.event-threads 4

   The default value for the event-threads parameters is 2.

The graph below provides the performance of small file creations and reads for RHGS 3.1, along with comparisons to some previous releases. The benchmark used for these tests is *smallfile*. The configuration used for tests in this section is as described below:

- 4 servers, each with a single brick
- brick configured on 12 disk RAID-6 device
- 2x2 distributed replicated volume
- 4 clients, fuse mounting the volume
- 10GbE network



The graph has results for RHGS 3.1 on both RHEL 6.7 and RHEL 7.1, since RHGS 3.1 is supported on both these platforms. Earlier releases of RHGS were supported only on RHEL 6. The “3.1 default” bar corresponds to RHGS 3.1 results on RHEL 6.7 with tunable parameters cluster.lookup-optimize, client.event-threads and server.event-threads at their default values. The “3.1 tuned” bar is for the case where these parameters have been tuned as per the recommendation given in this section. Similarly, the “3.1 EL7, default” and “3.1 EL7 Tuned” bars correspond to results on RHEL 7.1 with the parameters in their default and tuned state, respectively.

Analysis: The results show that performance for small-file creates and reads has improved significantly since RHS 3.0.3. The results also show that turning on the *cluster.lookup-optimize* parameter has a beneficial effect on create performance. Finally, the wide range of improvements made in RHEL 7 results in better performance compared to RHEL 6 on these tests.

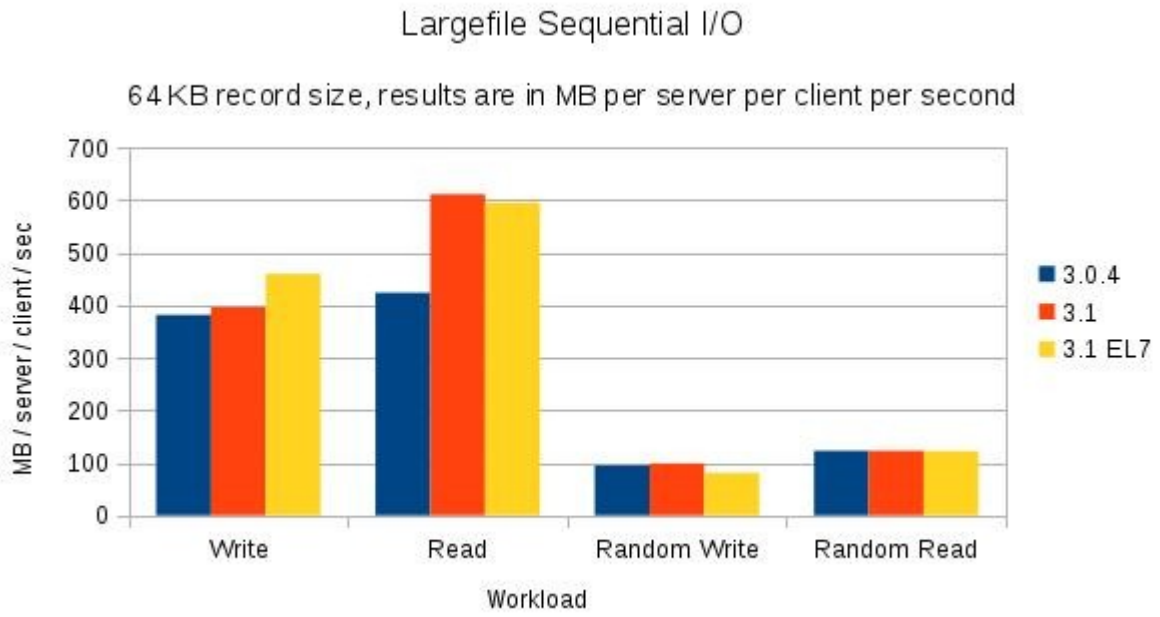
2.2 Large-file Performance

Performance for large-file use-cases has traditionally been a strength for RHGS. The RHGS 3.1 release features further improvements in this area as a result of system tunable parameter changes in the redhat-storage-server package.

The graph below shows a performance comparison of RHGS 3.1 versus RHGS 3.0.4 for a large-file workload. The benchmark used for these tests is *iozone*, which was run in distributed mode across multiple clients.

The configuration used for tests in this section is as described below:

- 4 servers, each with a single brick
- brick configured on 12 disk RAID-6 device
- 2x2 distributed replicated volume
- 4 clients, fuse mounting the volume
- 10GbE network
- 4 iozone threads on each client, each thread creating an 8GB file



RHGS 3.1 results are reported for both RHEL 6.7 and RHEL 7.1, with RHEL 7.1 results labeled “3.1 EL 7”. The results show a marked improvement in sequential read performance in RHGS 3.1 compared to 3.0.4.

3. Performance of New Features

This section analyzes the performance of features that are new in RHGS 3.1 and have a significant performance impact. These include:

- Erasure-coding
- NFS access with NFS-Ganesha
- BitRot Detection

3.1 Erasure Coding

The most significant new feature in RHGS 3.1 is the introduction of distributed erasure-coding (EC) storage in support of *large-file sequential workloads*. EC provides several benefits:

- reducing the ratio of physical-to-usable storage by a factor of almost 2
- reducing network load, resulting in better network utilization on writes
- breaking dependency on hardware RAID6 resulting in lower-cost configurations and equivalent or higher data durability in some cases

For large-file sequential workloads, we see throughput equivalent to traditional RAID6 2-replica configurations of RHGS v2 and for writes, better throughput than with JBOD replica 3 configurations.

Warning: At this time, we do not recommend use of EC storage for workloads where predominant I/O pattern is random I/O or small-file I/O. Why: for random I/O and small files, EC requires many physical I/O requests per logical I/O request.

Warning: At this time, we do not recommend use of EC storage for workloads where response time is critical, such as transactional workloads. Why: EC spreads an individual I/O request across hosts and response time will be determined by the slowest response time in the bricks within the EC subvolume.

Note: See documentation for data durability configuration guidelines, which MUST be followed in order for Gluster to provide high-availability and durability in line with user expectations.

The EC configurations supported with this release are EC4+2, EC8+3 and EC8+4.
Listed below are the test cases that were performed to evaluate the performance of EC4+2, EC8+3 and EC8+4 for both small files and large files in RHEL Server 7.1 environment:

For Small file Performance we performed:

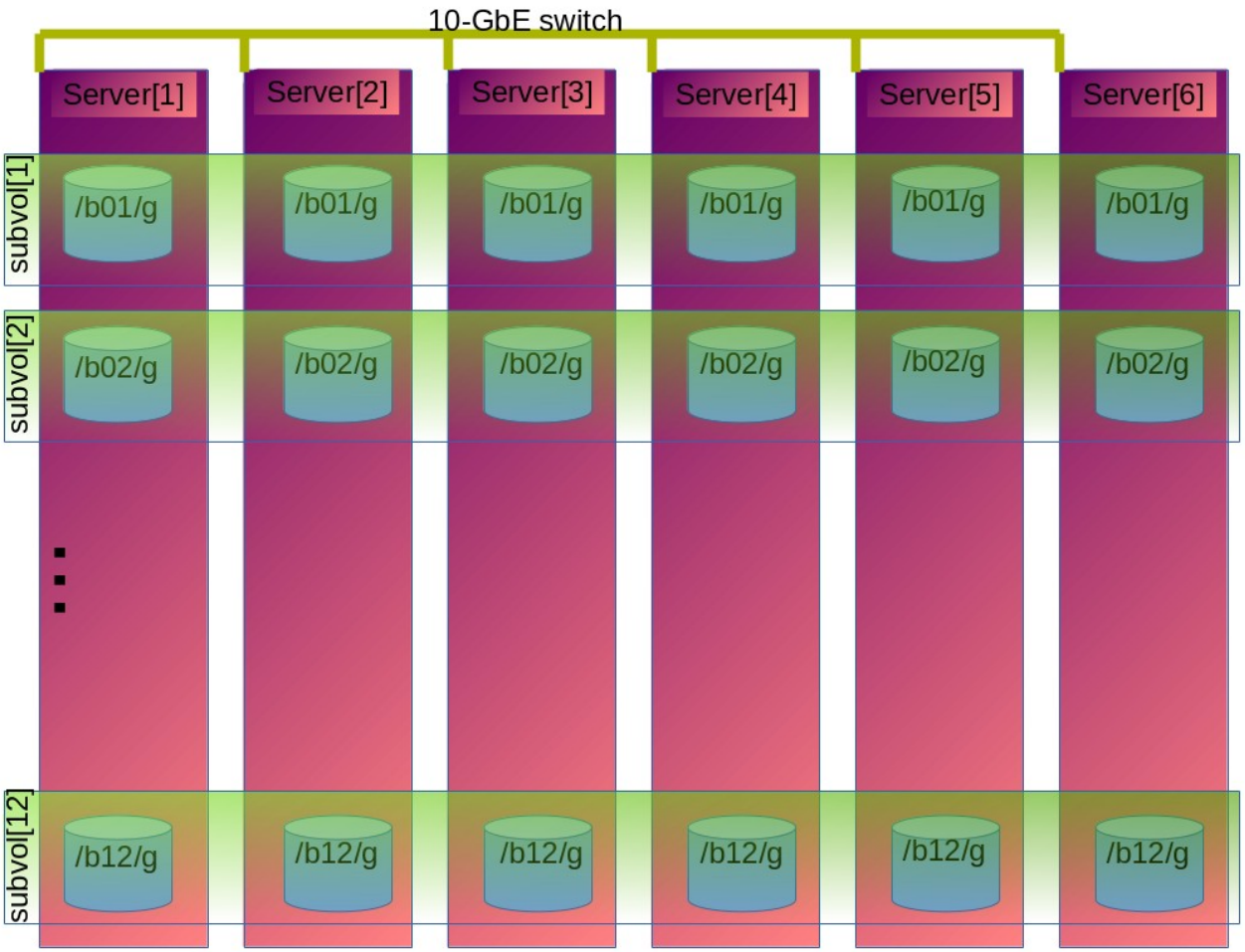
- Create-Read Performance Comparison between Erasure Coded 8+4 volume and Replica 3 volume.
- Small file performance comparison between different erasure coding configurations.

For Large File Performance we performed:

- Large file sequential and random IO performance comparison between Erasure Coded 8+4 volume and Replica 3 volume.
- Large file sequential and random IO performance comparison between different erasure coding configurations.

To test EC performance, we created an Erasure Coded Gluster volume across 6 servers, each with 12 10K RPM SAS drives in a JBOD configuration, with 1 10-GbE NIC port/server. There were 6 clients working as load generators.

The diagram and explanation below shows the layout of sub-volumes for EC4+2 bricks in 6 servers that were used for erasure code testing. (Note: Similar process was used to set up EC8+3 and EC8+4 gluster volumes)



The following script was used to create the Gluster EC4+2 volume, only instead of server[1] through server[6] we use the test lab hostnames gprfs0{13,14,15,16,22,24}. Each brick, represented by a blue disk shape in the above diagram, is a single disk drive (example: **/dev/sdc**) formatted with a thin LVM volume and XFS filesystem (example: **/bricks/b02**), and a subdirectory **g/** is created for the Gluster brick.

```
gluster v create ec42 disperse 6 redundancy 2 \
gprfs0{13,14,15,16,22,24}-10ge:/bricks/b01/g

for n in `seq -f "%02g" 2 12` ; do
     gluster volume add-brick ec42 \
          gprfs0{13,14,15,16,22,24}-10ge:/bricks/b$n/g
done
gluster v start ec42
```

The Gluster volume is started only after all bricks have been added to it.
To make it easier to understand, we add the first EC subvolume (first row in the diagram) in the **gluster volume create** command, and the add subsequent EC subvolumes (subsequent rows in diagram) using the **gluster volume add-brick** command.

EC4+2 means that we can lose any two bricks of a subvolume and still access the data in that subvolume. This is similar to how RAID6 behaves, except that RAID6 volumes were entirely contained in a single host, whereas EC4+2 volumes are spread across hosts. This means up to 2 hosts can go down without loss/unavailability of data in this configuration, since there is only 1 brick/hosts. Similarly EC8+3 means that we can lose any three bricks of a subvolume and still access the data in that subvolume and EC8+4 means that we can lose any four bricks of a subvolume and still access the data in that subvolume.

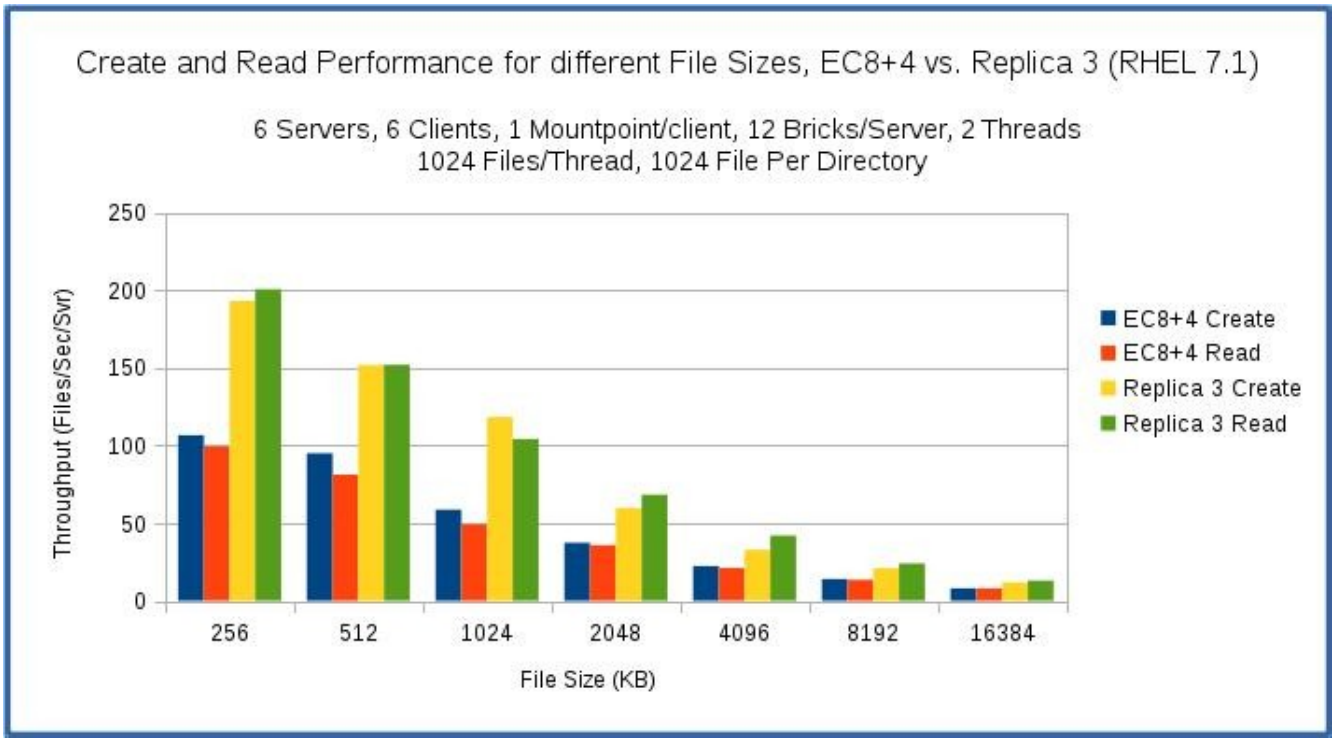
3.1.1 Small File Performance

The *smallfile* benchmark was used to analyze the performance of small files in various erasure coded volume configurations.

3.1.1.1 Comparison between EC8+4 and Replica 3 Volume

The below graph compares the Create and Read performance of EC8+4 volume and Replica-3 volume (24x3 distribute-replicate volume with bricks in JBOD configuration) for different small file sizes in RHEL 7.1 configuration. These two configurations use all 72 drives available across all 6 servers and hence they were chosen for comparison.The throughput is in files/sec/server.

3.1.1.2



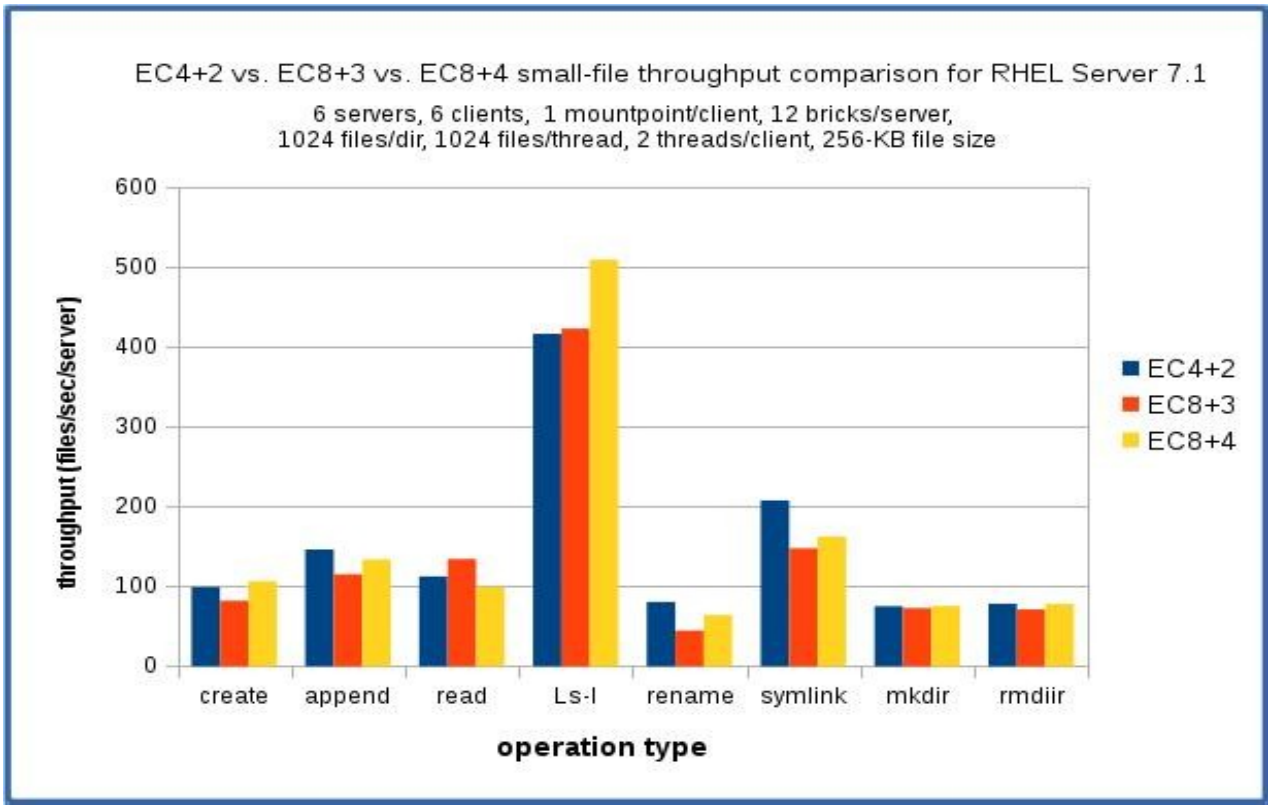
Analysis:

- As file size is increased from 256KB to 16MB the number of files per server can handle reduces.
- For create operation EC8+4 configuration achieves 50% to 68% of Replica 3 performance for different file sizes tested.

- For read operation EC8+4 configuration achieves 47% to 62% of Replica 3 performance for different file sizes tested.
- EC is at a significant disadvantage compared to replica-3 because of the number of bricks that have to be touched for every small file that is written, but this disadvantage shrinks as file size increases.

3.1.1.3 Comparison of different Erasure Coded configurations

The following graph shows the performance comparison of EC4+2, EC8+3 and EC8+4 configurations when different small file operations were performed in RHEL server 7.1 setup. Here EC4+2 and EC8+3 configuration are using 66 backend drives whereas EC8+4 is using 72 backend drives.



Analysis:

- For small file create operation, EC8+4 performed 7% better than EC4+2 and about 22% better than EC8+3.
- For small file append operation, EC4+2 performed about 8.5% better than EC8+4 and about 22% better than EC8+3.
- For small file read operation, EC8+3 performed about 16% better than EC4+2 and about 26% better than EC8+4ion
- For small file ls-l operation, EC8+4 outperformed both EC4+2 and EC8+3 by about 18%.
- For small file operations rename, symlink EC4+2 outperformed both EC8+3 and EC8+4 by 20% to 43%.
- For small file operations mkdir, rmdir all EC configuration gave comparable

performance.

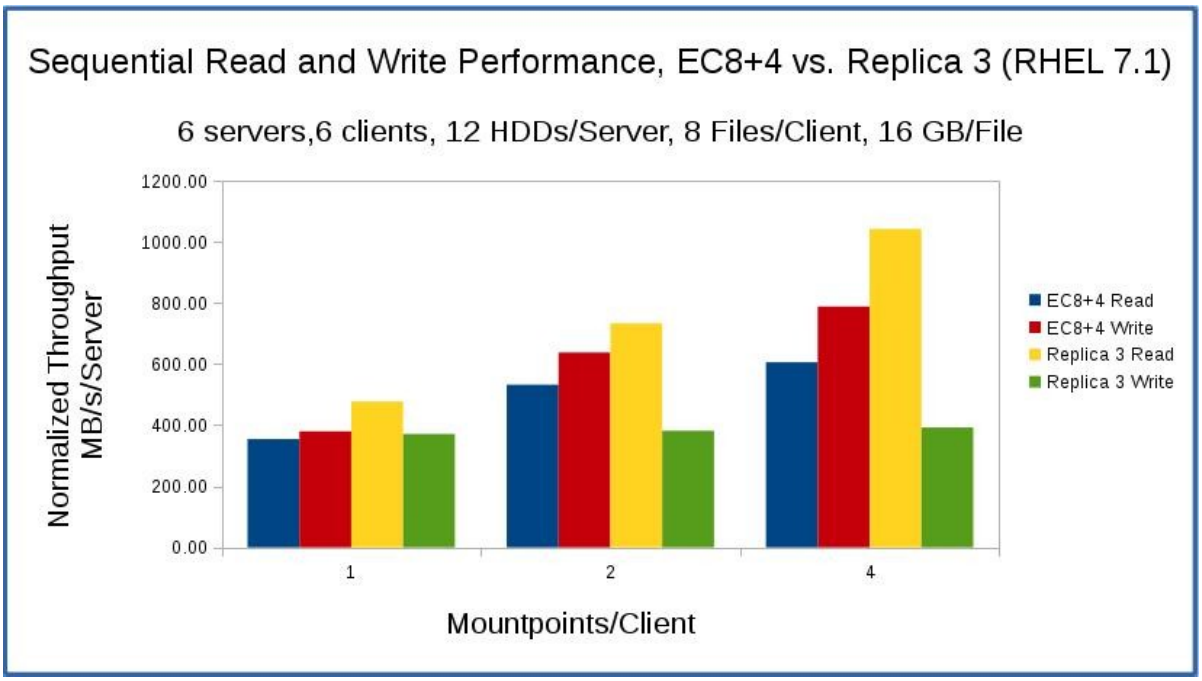
3.1.2 Large-file Performance

FIO benchmark was used to analyze the performance of large files in different erasure coded volumes. For testing purposes, we have used between 1 and 4 mount points on each client to simulate different ratios of Gluster native clients per server.

3.1.2.1 Comparison between EC8+4 and Replica 3 Volume

3.1.2.1.1 Sequential I/O Workload performance

The below graph compares the read and write performance of an EC8+4 volume and Distributed Replica 3 Volume for sequential I/O workload in RHEL 7.1 configuration. These two configurations use all 72 drives available across all 6 servers and hence they were chosen for comparison.The throughput is in MB/sec/server.



Analysis:

- For sequential read workload Replica 3 volume almost performed at line speed, giving a throughput ~1000MB/sec/svr.
- For sequential read workload, EC8+4 achieves 58% to 74% of Replica-3 performance for different mountpoints/client tested.
- For sequential write workload, EC8+4 achieves 102% to 201% of Replica-3 performance for different mountpoints/client tested.
- Problem with EC reads is the small chunk size and it needs to synchronize with each host to read each stripe. This reduces the performance.
- For sequential writes EC8+4 performed about 2% to 50% than replica 3 volume.

Replica 3 writes were limited by network bandwidth.

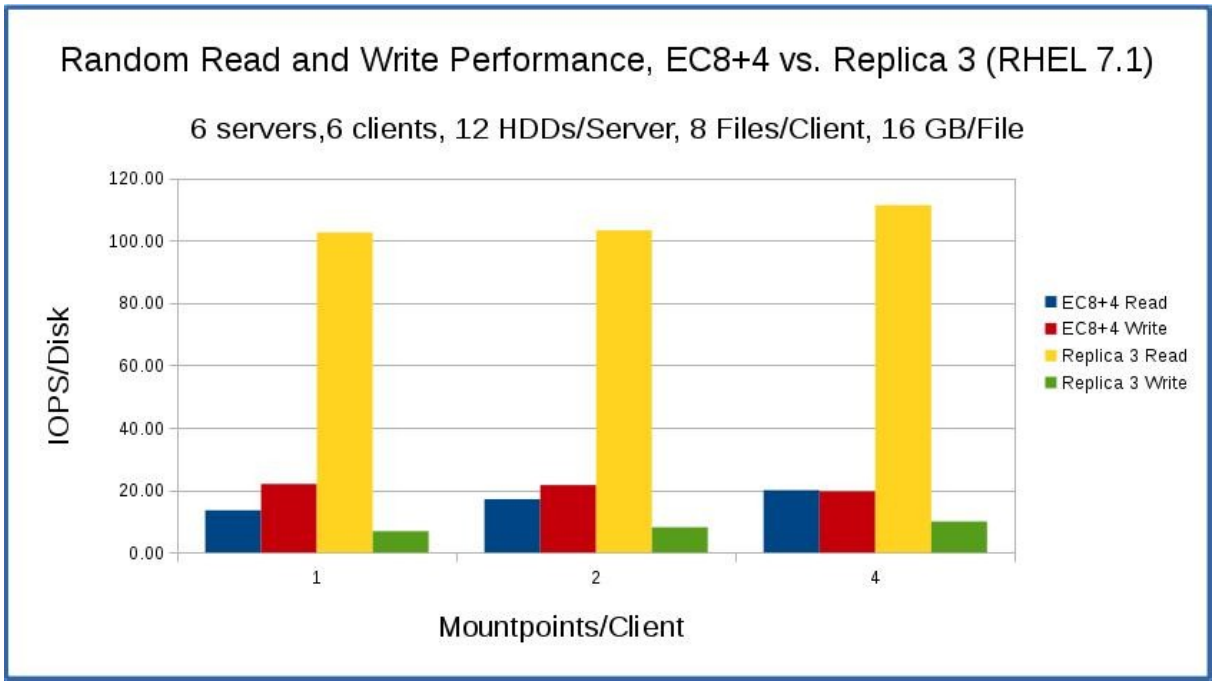
Throughput for large-file performance is displayed using the metric MB/s/server, because often a sequential distributed workload is bottlenecked at the network layer or by the max transfer rate of the storage controller, so this metric gives us an idea of what we would expect for a good result - in this case, 10-GbE line speed is the best we can hope for.

For Gluster EC implementation, both the data and redundancy blocks are transferred, on both reads and writes. The theoretical network throughput limit is 10 Gbit/sec ~= 1200 MB/s, and since redundancy data is ⅓ of the traffic, the throughput limit is then 1200 MB/s x ⅔ = 800 MB/s (top of the Y axis). What we observe is that throughput in this case with 4 mount points per client, we do get close to this network limit on writes, and within 25% of this limit on reads.

For sequential writes with 2 way replication, the theoretical limit for write throughput would be 600 MB/s, and the theoretical limit for 3-way replication would be 400 MB/s. For sequential reads, traditional replication only reads 1 copy of the data so the theoretical limit would be 1200 MB/s. In practice most tests see closer to 600 MB/s because of various other effects. In summary, we are doing as good or better than traditional replication for sequential writes, with a 30% reduction in sequential reads. Keep in mind the other benefits of erasure coding that we obtain while doing this.

3.1.2.1.2 Random I/O Workload performance

The below graph compares the write and read performance of EC8+4 volume and Distributed Replica 3 Volume for random I/O workload in RHEL 7.1 configuration. These two configurations use all 72 drives available across all 6 servers and hence they were chosen for comparison.



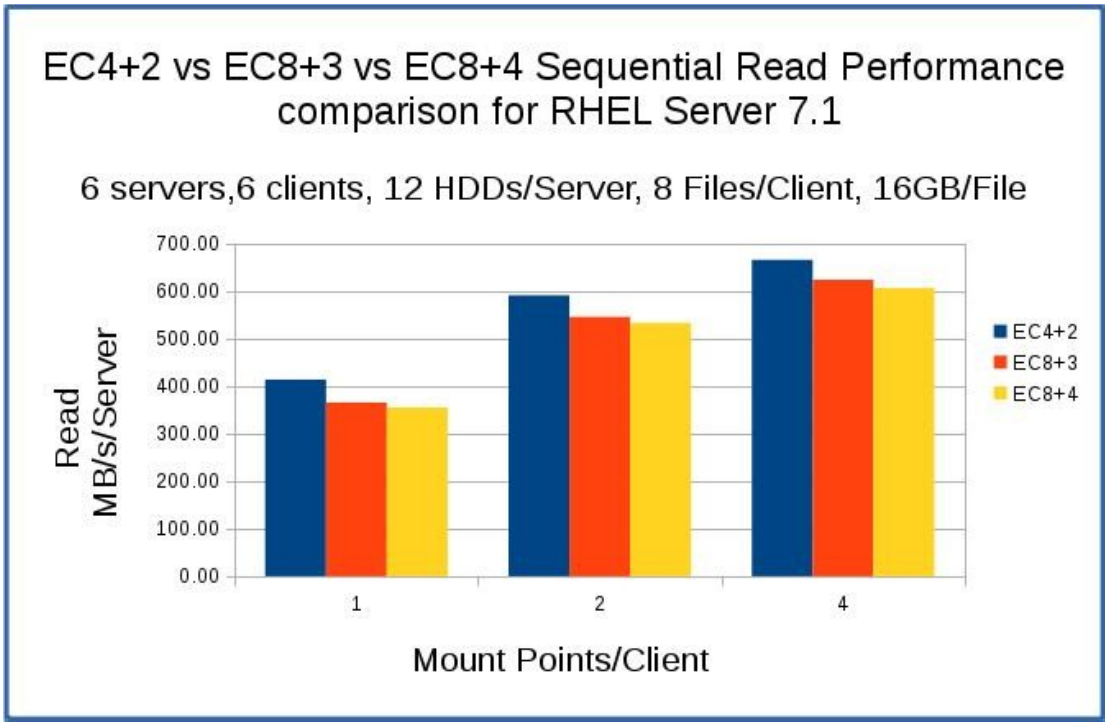
Analysis:

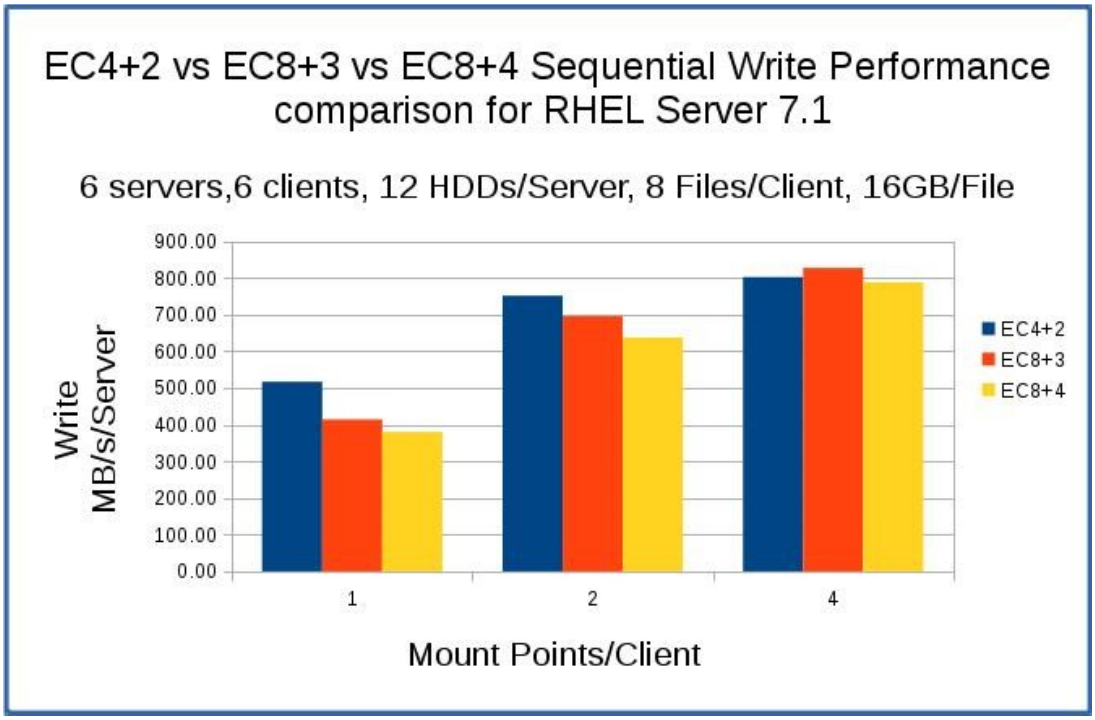
- For random read workload, EC8+4 achieves only 14% to 18% of Replica-3 performance for different mountpoints/client tested.
- For random write workload, although EC8+4 achieves 196% to 322% of Replica-3 performance for different mountpoints/client tested but the best random write performance achieved is ~22 IOPS/Disk .
- The result shows the EC is not good for Random workload performance.

3.1.2.2 Comparison of different EC configurations

3.1.2.2.1 Sequential I/O Workload

The following graphs show the performance comparison amongst EC4+2, EC8+3 and EC8+4 for large file sequential read and write performance in RHEL server 7.1 setup.



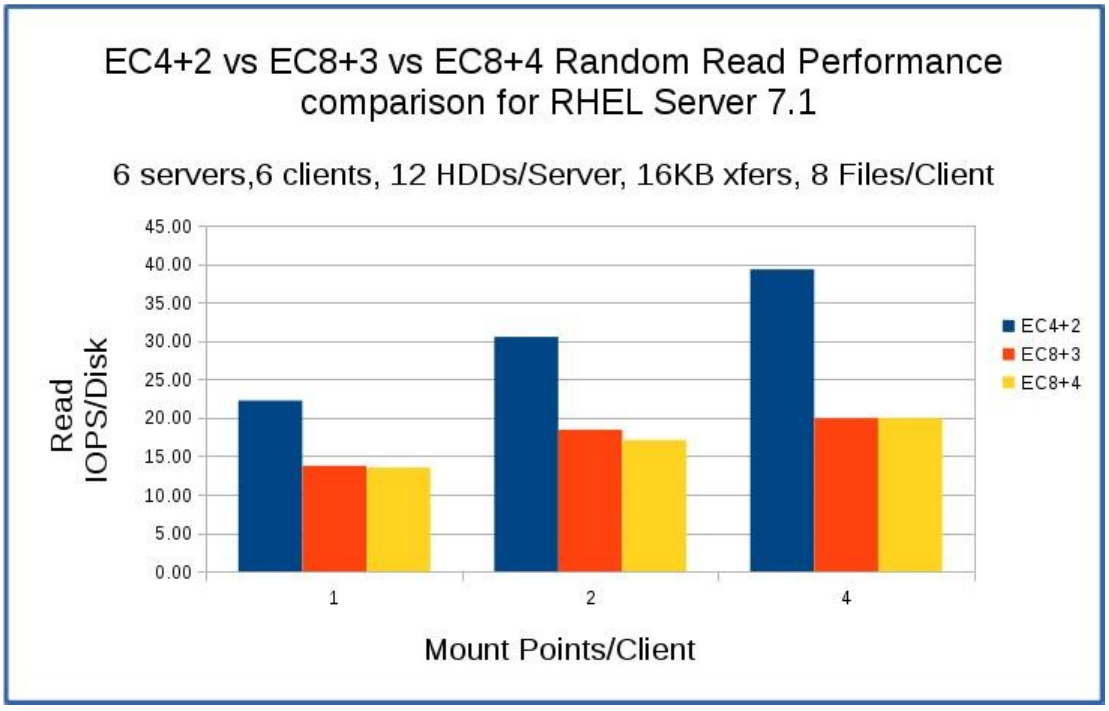


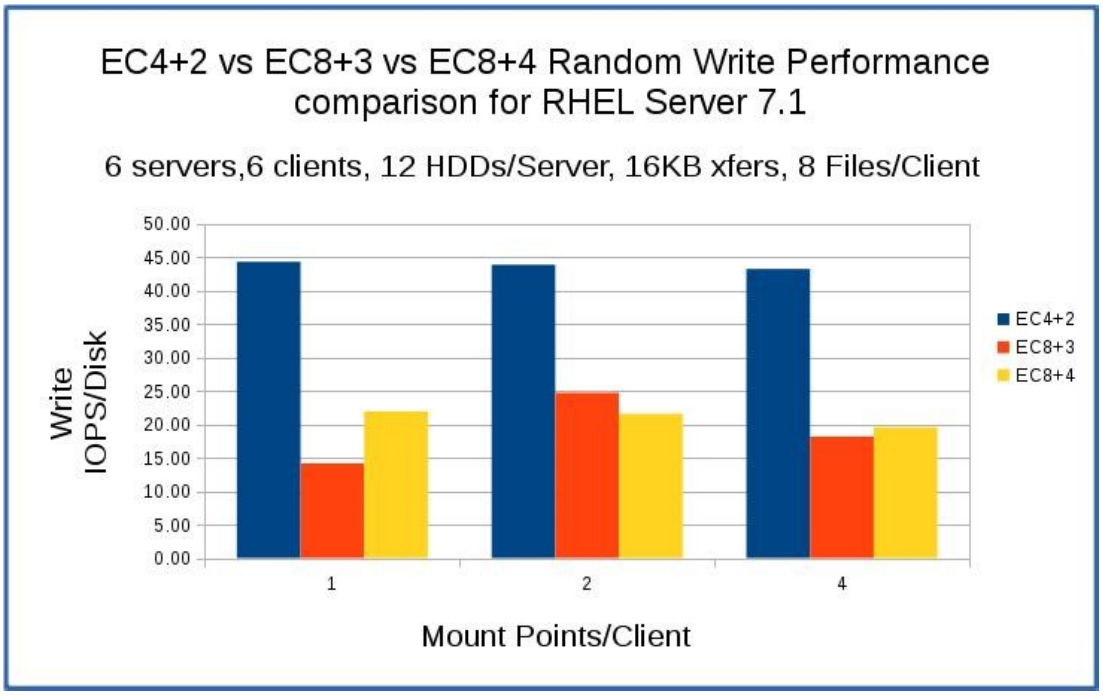
Analysis:

- For sequential large file read workload EC4+2 performed better than EC8+3 and EC8+4 for all mount points tested. EC4+2 performance was about 9% to 14% better across different mount points.
- The maximum read throughput achieved per server with EC4+2 was ~664 MB/Sec.
- For sequential large file write workload EC4+2 performed better than EC8+3 and EC8+4 by about 15% to 26% for 1 and 2 mount points test. For 4 mount points test all erasure coded volume performed equally well.
- The maximum write throughput achieved per server was 827 MB/s. This was achieved with EC8+3 configuration with 4 mount points setup.

3.1.2.2.2 Random I/O Workload

The following graphs show the performance comparison between EC4+2, EC8+3 and EC8+4 for large-file sequential read and write performance in RHEL server 7.1 setup.





Analysis:

- Compared to large-file sequential I/O workload, large-file random I/O workload doesn't perform that well.
- EC8+3 and EC8+4 performance is almost half than EC4+2 for both random read and random write workloads. This is because we are involving all disks in the target EC subvolume for each individual logical random read/write.

Tuning: The following Gluster volume parameters were used:

```
client.event-threads: 8
```

Network tuning consists of use of MTU=9000 jumbo frames. Improvements to the tuning framework has been done in RHEL7:

- **tuned** RPM is installed by default, with the throughput-performance tuned profile being active by default.
- more modern kernel with better support for bonding (teaming)

Methodology: Testing for large-file workloads was done using upstream fio, which now has support for multi-client testing using a shared filesystem like Glusterfs. Small-file testing was done using the *smallfile* tool. Methodology is described somewhat here. This includes automated tests to ensure that the 10-GbE network was performing at line speed on all active NIC ports. Multiple samples were obtained for each test and data was cross-checked using the **pbench** perf data collection tool, which is in process of being upstreamed now by the Red Hat perf. dept. In practice, there are a variety of existing tools that can be used for perf data collection, such as Perf. Co-Pilot, collectl, xabbix, etc. and we strongly recommend that you use such a tool to monitor your configuration.

3.2 NFS-Ganesha

RHGS 3.1 supports NFSv3 and NFSv4 access to gluster volumes via gluster's integration with NFS-Ganesha which is an open-source, user-space NFS server implementation. NFS-Ganesha uses a File System Abstraction Layer (FSAL) which allows it to integrate with and NFS-export different types of file systems. FSAL-gluster allows RHGS volumes to be exported via NFS-Ganesha.

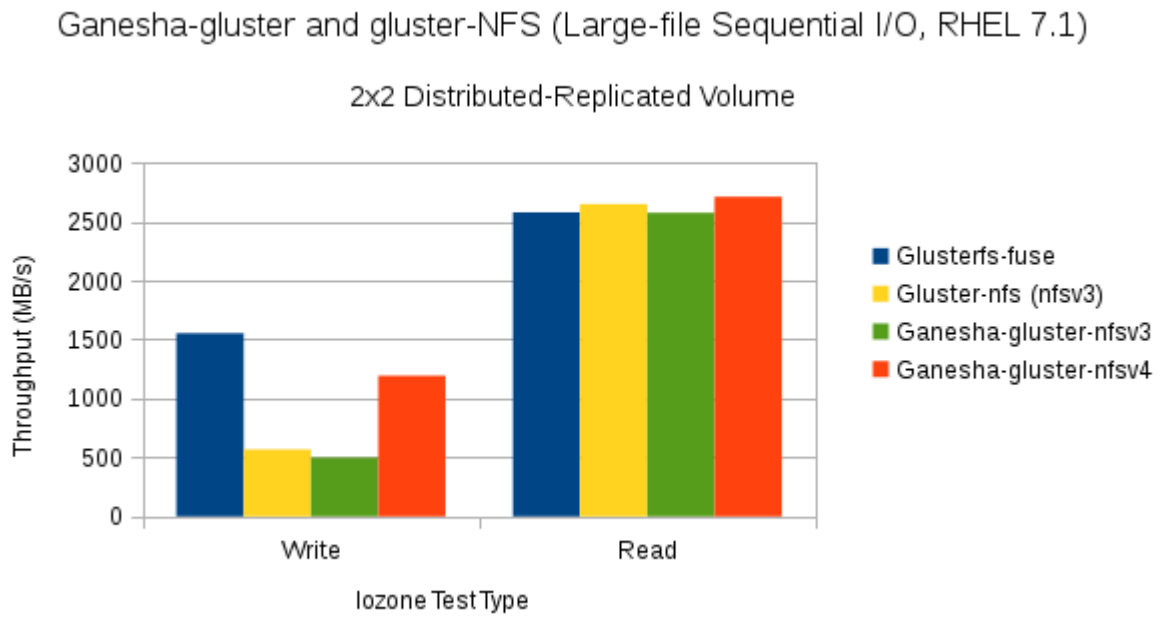
Prior to RHGS 3.1, RHGS only supported NFSv3 access through the Gluster native NFS implementation (gNFS).

Since gluster is a scale-out storage solution, it is critical that all accesses to gluster volumes not be funneled through a single NFS server. Hence, RHGS 3.1 supports multiple NFS-Ganesha heads simultaneously serving accesses to the same gluster volume, with transparent failover in the case of NFS head failure. This mode of operation requires that shared storage is configured in the form of a special gluster volume, named *gluster_shared_storage*, where all NFS-Ganesha heads can save their state. For the results reported here, *gluster_shared_storage* was configured as a replica-2 gluster volume.

3.2.1 NFS-Ganesha Large-file Sequential I/O Performance

The graph below compares performance of an RHGS volume accessed using different protocols: glusterfs-native-fuse, gNFS, Ganesha-gluster NFSv3 and Ganesha-gluster NFSv4. The workload is a large-file, sequential I/O workload generated using the iozone benchmark. For the graph below, the following applies:

- 4 clients, 4 servers (RHEL 7.1)
- single RAID-6 virtual disk per server, with 12 HDDs
- 10Gb Ethernet on all systems with jumbo frames, MTU at 9000
- 4 NFS heads in use, in the case of ganesha-gluster and gNFS, i.e. each server has an active NFS head
- each client mounting from a different NFS head
- 24 iozone threads (6 per client), total data set size 480g



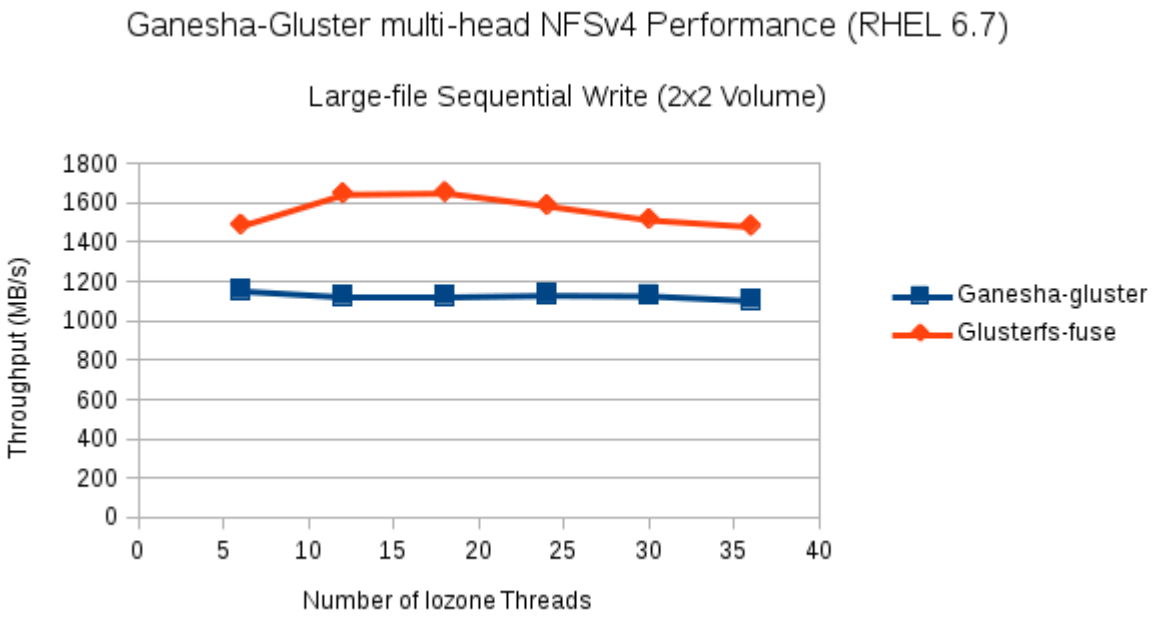
Ganesha-gluster NFSv3 in the above graph is matching the performance of gNFS, which is also NFSv3. Ganesha-gluster NFSv4 is doing significantly better on the write test with this workload.

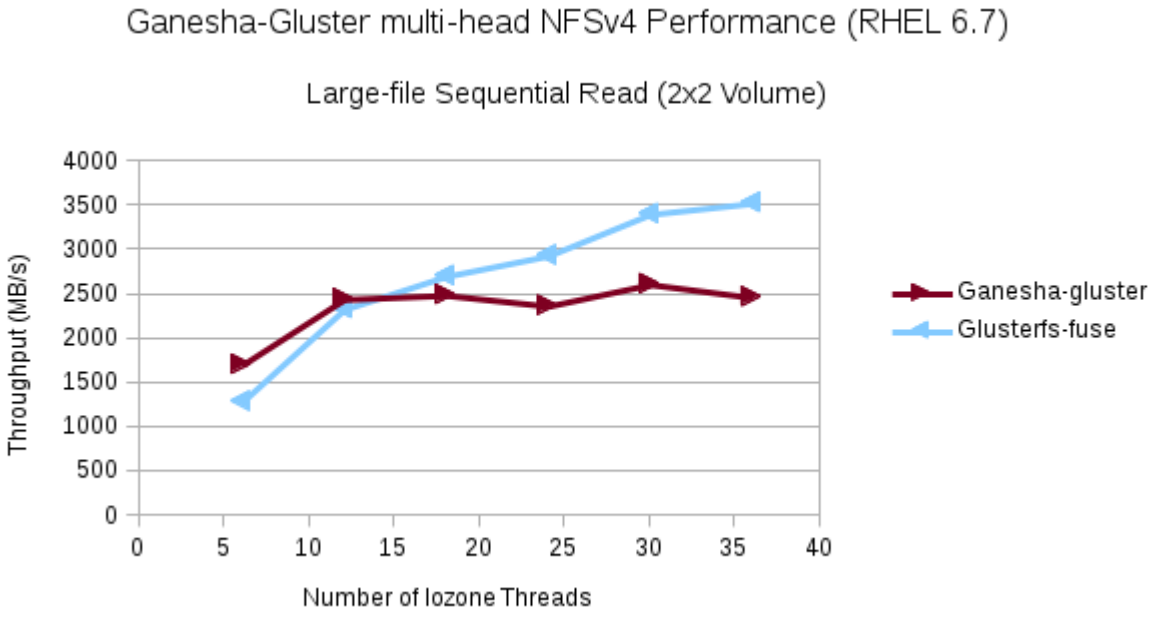
Analysis: NFS access to RHGS volumes involves a higher latency path for I/Os compared to gluster's native protocol. NFS requests are first sent to the NFS server (NFS-Ganesha server or gluster NFS process, in this comparison), which then needs to send it to the appropriate gluster brick server(s). For most workloads, this means that NFS access will be less efficient/slower than access using the gluster native protocol. However, for throughput-oriented workloads, some of the higher latency can be masked. In the read test above, the NFS variants are able to match gluster-native-fuse performance.

The graphs below shows a comparison between NFS-Ganesha and gluster native protocol (fuse mount) on the same workload (large-file, sequential I/O generated using iozone) with a varying number of threads. For the graphs below, the following applies:

- 6 clients, RHEL 7.1
- 4 servers, RHEL 6.7
- single RAID-6 virtual disk per server, with 12 HDDs
- 10Gb Ethernet on all systems with jumbo frames, MTU at 9000

- 4 NFS heads in use, in the case of ganesha-gluster
- NFSv4 mount
- 480 GB data set; file size obtained by dividing data set size by the total number of threads

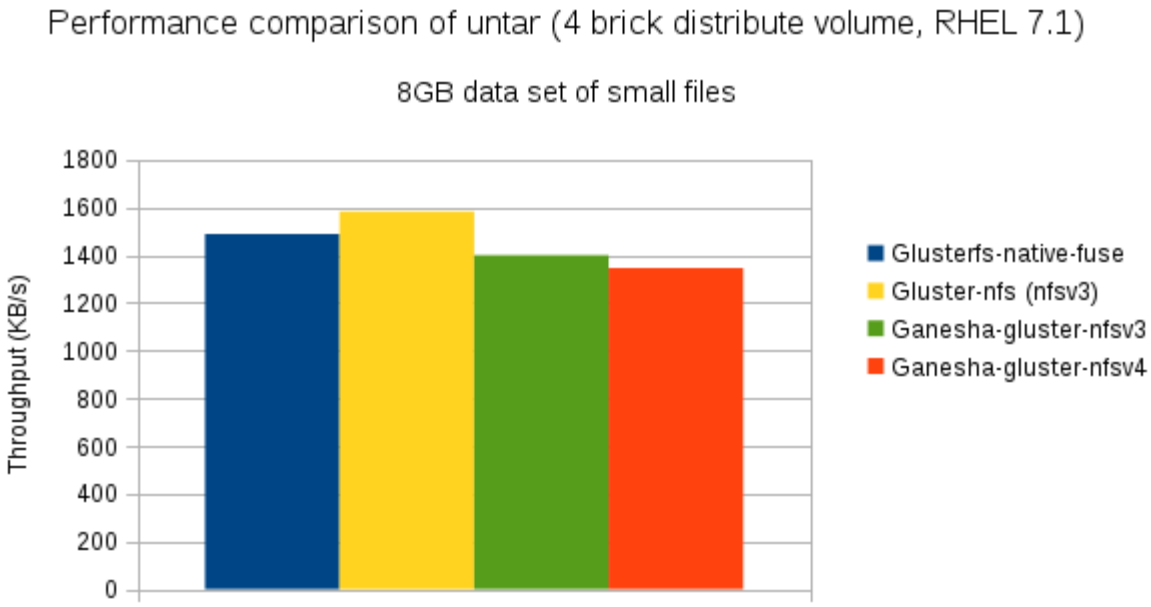




Analysis: In these graphs, there are 6 clients mounting from 4 NFS heads, so load is not evenly spread among the 4 NFS heads: 2 of the NFS heads serve 2 clients each, whereas the other 2 serve 1 client each. This, along with the extra hop needed for NFS requests, means that NFS-Ganesha trails gluster native protocol in performance.

3.2.2 NFS-Ganesha Small File Performance

The graph below compares the performance of NFS-Ganesha to other implementations on a single-threaded workload creating small files. The workload in this case is an untar of an 8GB data set of small-files of size 16KB-64KB.



This result again shows Ganesha-gluster close to gNFS in performance. This is significant because NFS-Ganesha is a more sophisticated implementation and provides a path to features like NFSv4 delegations and pNFS. The graph shows that it does not come with a significant performance penalty.

3.3 Bitrot Detection

RHGS 3.1 adds support for BitRot detection, which helps detect silent corruption of bits on storage media. BitRot detection works by calculating and storing checksums on files, and periodically scrubbing files to detect checksum mismatches which indicate corruption of bits.

The feature can be enabled or disabled using the command:

gluster volume bitrot <VOLNAME> {enable|disable}

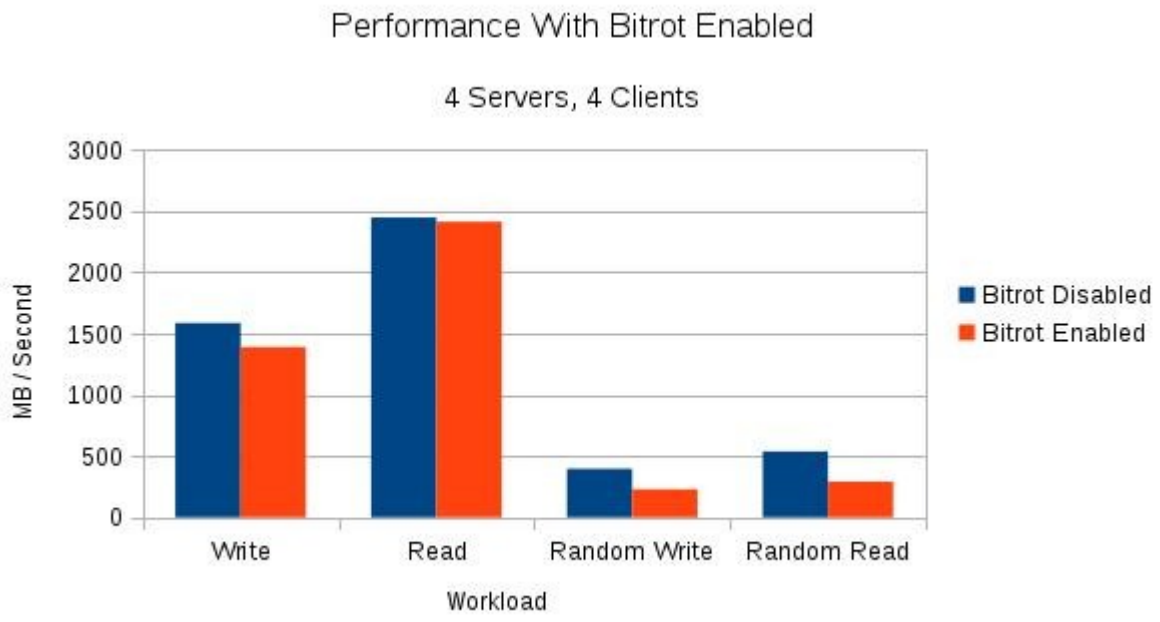
Checksum computation and scrubbing can potentially have a significant impact on performance. The BitRot detection feature provides tunable parameters to control its impact on application performance. These are listed below:

```
gluster volume bitrot <volname> scrub-throttle {lazy|normal|aggressive}
gluster volume bitrot <volname> scrub-frequency
{hourly|daily|weekly|biweekly|monthly}
gluster volume set <volname> features.expiry-timeout <seconds>
```

The parameter *features.expiry-timeout* determines how soon after a change to a file the checksum is computed, and it defaults to 120 seconds. The default value for the *scrub-throttle* parameter is *lazy*, and this mode reduces the performance impact of scrubbing. When the *scrub-throttle* parameter is set to *normal* or *aggressive* the performance impact of scrubbing increases.

The graph below characterizes the performance impact of BitRot detection by comparing performance with BitRot enabled against a baseline with BitRot disabled. The benchmark used is *iozone* in distributed mode. The configuration used for the tests is as described below:

- 4 servers, each with a single brick
- brick configured on 12 disk RAID-6 device
- 2x2 distributed replicated volume
- 4 clients, fuse mounting the volume
- 10GbE network
- 4 iozone threads on each client, each thread creating an 8GB file



The graph shows a small performance hit for sequential workloads with bitrot enabled, within 10% of the case where BitRot is disabled. On random reads we see a drop of 45%. Random

Writes see a drop of 42%. Note that these tests were run with *scrub-throttle* set to the default i.e. *lazy*. Using *normal* or *aggressive* as the *scrub-throttle* setting will increase the performance hit. There are plans to update the throttling implementation to reduce the performance hit from BitRot Detection.

4. Performance of Legacy / Refactored Components

This section analyzes performance of some existing features which were enhanced in the RHGS 3.1 release. These include:

- Rebalance
- Quota
- Samba

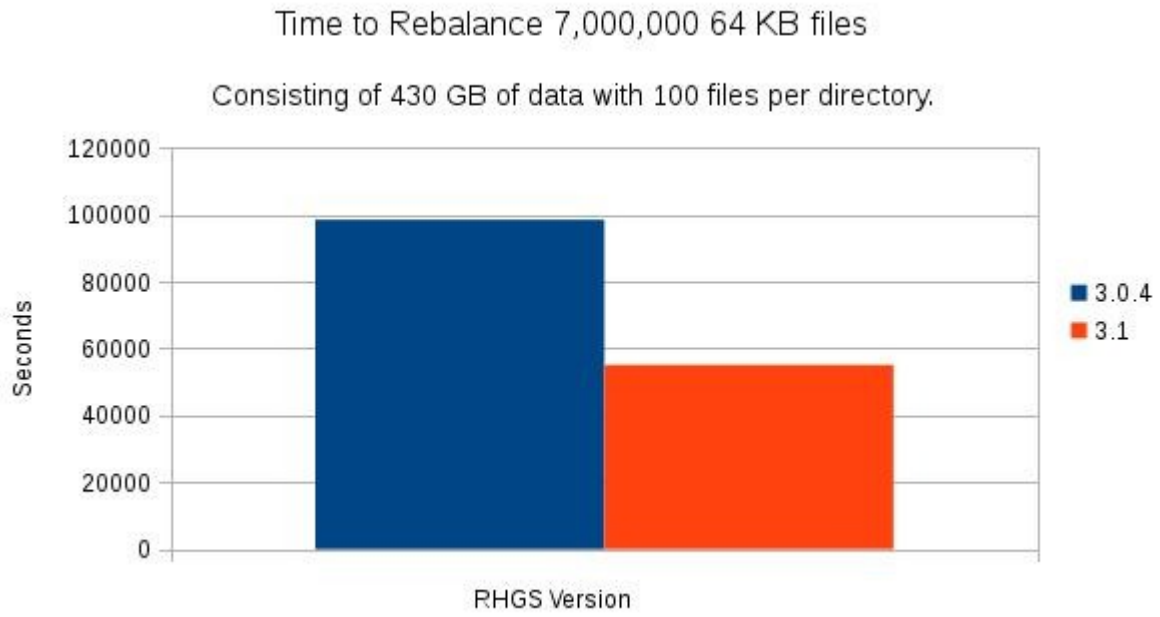
4.1 Rebalance Performance

Rebalance is a critical feature in RHGS which enables incremental scaling of deployments by migrating and balancing files across servers as servers are added or removed. In the RHGS 3.1 release, the rebalance implementation was enhanced to allow multiple migrations to happen in parallel per server, thereby allowing rebalance to complete faster. The implementation was also enhanced to allow control over the impact of rebalance on application performance through the tunable parameter listed below:

gluster volume set <testvol> cluster.rebal-throttle <lazy|normal|aggressive>

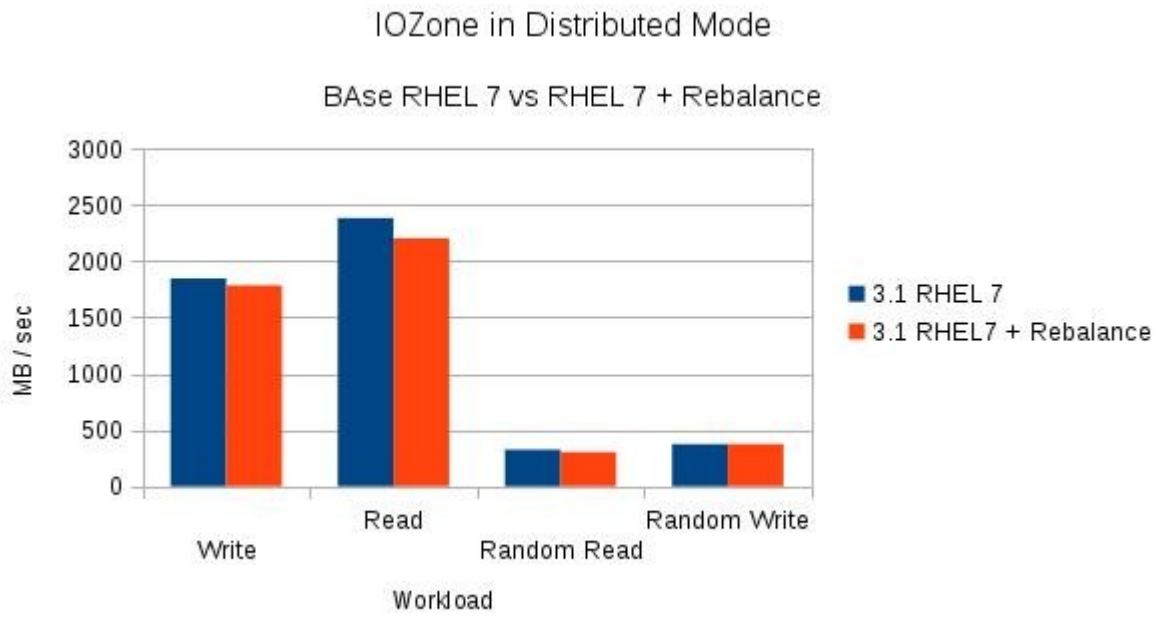
The default value of the *rebal-throttle* parameter is *normal*. Setting *rebal-throttle* to *aggressive* will generally cause rebalance to complete faster by allowing more parallelism in data migration, but will have a higher impact on application performance. Setting *rebal-throttle* to *lazy* will cause rebalance to be slower because data migration will be limited to a single thread per server, but this setting will reduce impact on application performance.

In the graph below rebalance was run with the default setting i.e. with *rebal-throttle* set to *normal*. The data set on which rebalance was performed was created using the *smallfile* benchmark, and consisted of 7,060,700 files each 64KB in size, with 100 files per directory for a total data set size of 430 GB. The data set was created on a 4 node 2x2 distributed-replicated volume. Rebalance was initiated after adding 2 new nodes to the volume to make it a 3x2 volume. The graph compares RHGS 3.1 and RHGS 3.0.4 on the time taken for rebalance to complete.



The graphs shows a significant reduction in time taken for rebalance to complete with RHGS 3.1, about 45% less compared to RHGS 3.0.4.

Another aspect of rebalance is its impact on application I/O performance. The graph below measures this impact by running a distributed IOZone benchmark with and without rebalance in progress. For the case where rebalance was in progress, a data set with 430 GB of small files was being rebalanced. Note that the test below was on a 2x2 distributed-replicated volume on 4 servers running RHEL 7.



The results show only a small performance impact with rebalance in the default mode. Expect this to be greater when rebalance is running in *aggressive* mode, less when run in *lazy* mode.

4.2 Quota Performance

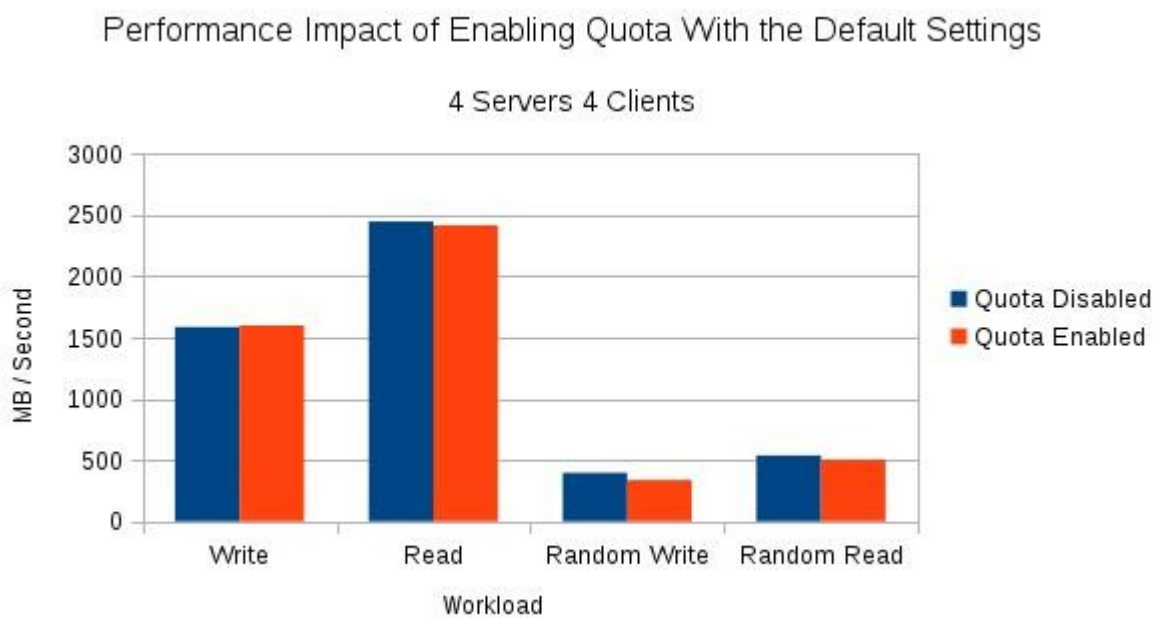
The implementation of the quota feature was reworked in the RHGS 3.1 release. However, there are no changes to the command-line. The command is as given below:

volume quota <VOLNAME> {alert-time|soft-timeout|hard-timeout} {<time>}

Volume quotas can potentially affect performance because the quota daemon has to perform accounting every hard-timeout and soft-timeout seconds. This accounting creates overhead and if the limit is passed will return an EDQUOT to the writing process or log that the soft limit has been passed in messages / the brick logs. Note that the hard-timeout setting controls how many seconds between accounting checks of the quota daemon and the quota can be exceeded by the number of MB / second a client writes at multiplied by the number of hard-timeout seconds.

The graph below characterizes the performance impact of the quota feature by comparing performance with quota enabled against a baseline with quota disabled. The benchmark used is *iozone* in distributed mode. The configuration used for the tests is as described below:

- 4 servers, each with a single brick
- brick configured on 12 disk RAID-6 device
- 2x2 distributed replicated volume
- 4 clients, fuse mounting the volume
- 10GbE network
- 4 iozone threads on each client, each thread creating an 8GB file

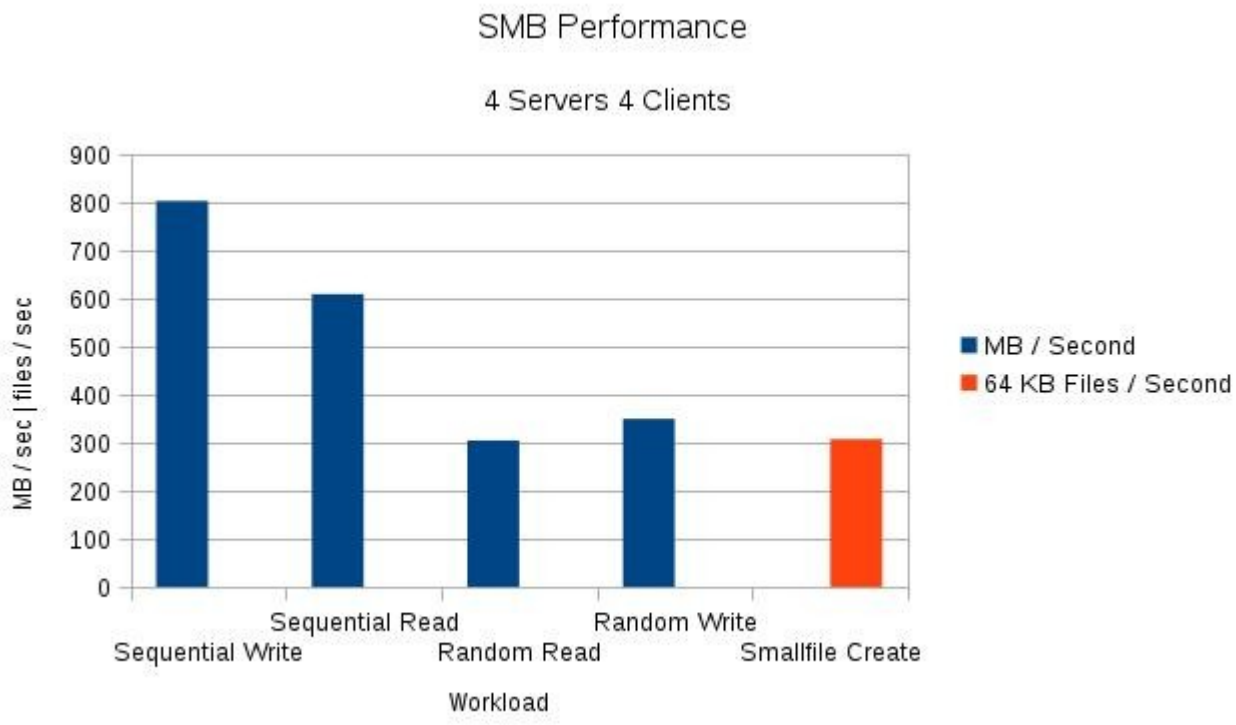


With the default settings, quota has little to no impact on sequential and random I/O, just like the quota implementation in RHGS 3.0.4. Note that as the hard-timeout timeout value is reduced, the performance impact of quota will increase but quota enforcement will be more strict.

4.3 Samba Performance

Early releases of RHGS supported SMB by having the Samba server re-export a locally mounted glusterfs mount point. However, in recent releases, including RHGS 3.1, SMB has been supported by integrating with libgfapi. Libgfapi is a library which provides an API for accessing data in GlusterFS. The libgfapi-based implementation has greatly improved SMB performance with large-file sequential and random IO.

The graph below shows the performance of large-file sequential and random IO as well as small-file creates on a 4 node 2x2 distributed-replicated (one twelve disk RAID 6 per node) volume mounted over SMB using 10GbE network. Each client mounted a different server. The benchmarks *smallfile* and *IOZone* were used for the small-file and large-file tests, respectively.


SMB Performance
4 Servers 4 Clients
MB / sec | files / sec
900
800
700
600
500
400
300
200
100
0
Sequential Write
Sequential Read
Random Read
Random Write
Smallfile Create
Workload
MB / Second
64 KB Files / Second

Appendix A: Revision History

Revision 1.0	October 6, 2015	John Harrigan
Initial Release		