



## **Red Hat Performance Briefs**

# **Red Hat Enterprise Virtualization Performance on Red Hat Storage**

**Performance Engineering**

**Version 1.0**

**May 2013**





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

Linux is a registered trademark of Linus Torvalds. Red Hat, Red Hat Enterprise Linux and the Red Hat "Shadowman" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

Intel® and Xeon® are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

© 2013 by Red Hat, Inc. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

The information contained herein is subject to change without notice. Red Hat, Inc. shall not be liable for technical or editorial errors or omissions contained herein.

Distribution of modified versions of this document is prohibited without the explicit permission of Red Hat Inc.

Distribution of this work or derivative of this work in any standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from Red Hat Inc.

The GPG fingerprint of the [security@redhat.com](mailto: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



# Table of Contents

1 Executive Summary.....	1
2 Configuration.....	2
2.1 Connectivity.....	2
2.2 Hardware.....	2
2.3 Software.....	3
3 Workloads .....	4
4 Results.....	6
4.1 Tuning.....	6
4.2 Scalability.....	10
4.3 Scaling Hosts.....	15
4.3.1 Large File I/O.....	15
4.3.2 Small File I/O.....	17
4.4 Detailed Performance.....	19
4.4.1 Large File I/O.....	19
4.4.2 Small File I/O.....	24
Appendix A: RHEV GUI displays.....	36
Appendix B: Gluster Volume Information.....	38
Appendix C: timed-par-for-all.sh.....	39



# 1 Executive Summary

In this set of tests, 128 RHEV (Red Hat Enterprise Virtualization) virtual machines (VMs) using Red Hat Storage (RHS) were used to characterize performance with varying numbers of hypervisors and Gluster servers. Each hypervisor and RHS server was connected through a network switch via a single 10-GbE NIC. This performance study can be used to:

- provide guidelines in recommendations regarding:
  - the ratio of hypervisors (hosts) to Gluster servers
  - the number of VMs per host
  - any host or VM tuning
- understand I/O bottlenecks
- compare basic workload local I/O to Gluster mount I/O in VM

**Conclusion:** In this case study, RHEV/RHS achieved near linear throughput as up to eight RHS servers and 16 RHEV hosts were added.

**Conclusion:** for small-file workloads, VM-local I/O has a significant performance advantage over I/O through a Gluster mount, even when the VM-local I/O is to a virtual disk image stored within the same Gluster volume.

There are some important differences between the VMs used in this study and more “real world” applications:

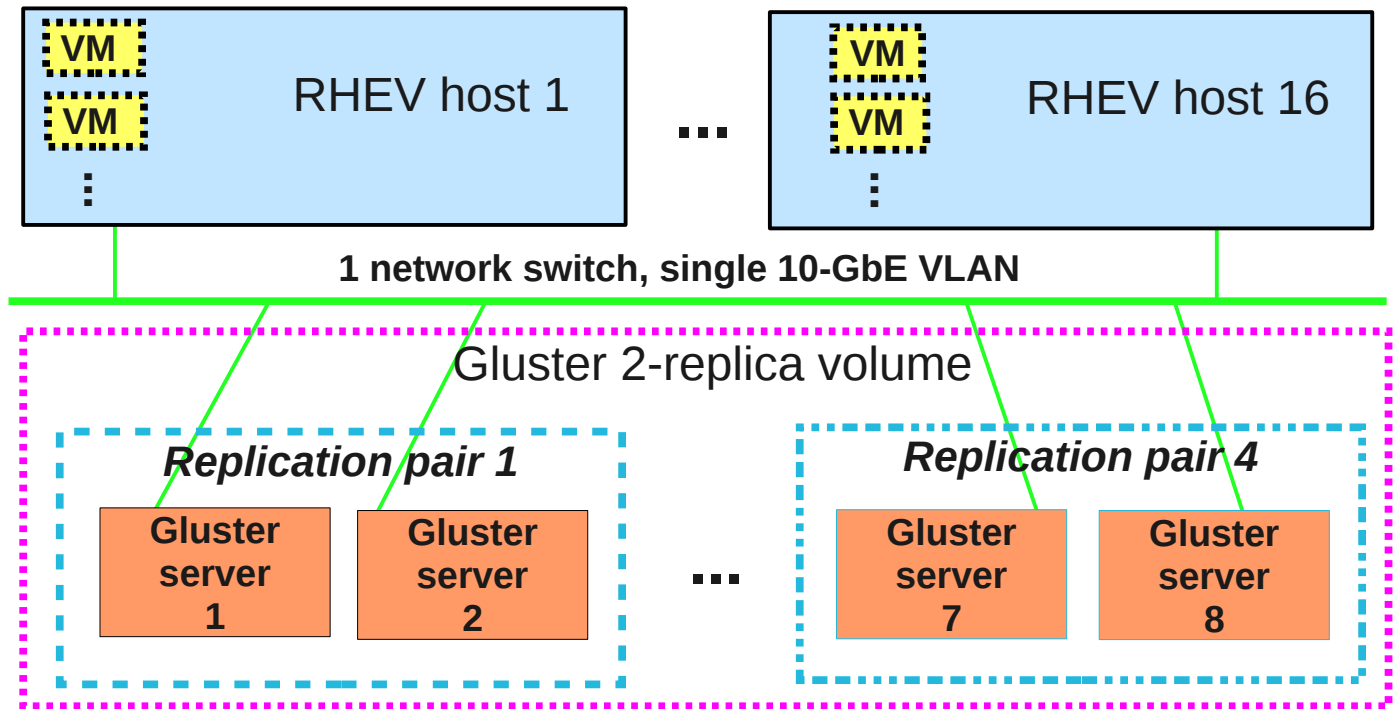
- *All VMs were executing an I/O workload 100% of the time* -- As such the results seen here should be interpreted accordingly. For example, to support 1000 VMs that may only be doing I/O to a Gluster volume for a small fraction of elapsed time on average, the VM density on Gluster storage could be proportionally increased provided peak loads are taken into consideration. The activity of VMs in an existing configuration can be measured to make these estimates.
- *VMs possessed only 500MB of RAM* -- VMs were sized to allow as many VMs as possible into the fewest hosts. This is not necessarily a good deployment model for more real-world configurations and in fact many applications could benefit from additional VM memory for cache-friendly applications such as software builds.
- *VMs possessed only one vCPU* -- this was also to allow a maximum number of VMs as possible into a single host in order to focus on VM I/O performance. However, such a low density of CPUs per VM would be inappropriate for applications such as Hadoop where there is a significant computational component.



## 2 Configuration

### 2.1 Connectivity

The figure below describes the physical connectivity of the hardware used in testing.



*Figure 2.1-1: RHEV on RHS Physical Configuration*

### 2.2 Hardware

<b>Servers (8)</b>	Dell PowerEdge R510, BIOS v 1.9.0 12core/24cpu Intel Xeon CPU X5650 @2.67 GHz 48 GB RAM 1x Intel 82599EB 10-Gigabit HBA 1x PERC H700 MegaRAID controller (12 drives in a RAID6 LUN with 256-KB stripe size) BIOS: hyperthreading and virtualization disabled, power management set to "OS control"
<b>Hosts (16) Virtual Machines (128) RHEV Management (1)</b>	Dell PowerEdge R510, BIOS v 1.9.0 12core/24cpu Intel Xeon CPU X5650 @2.67 GHz 48 GB RAM 1x Intel 82599EB 10-Gigabit HBA
<b>Storage</b>	Cisco Nexus 7010 switch, 4x 48-port line cards Single VLAN using Jumbo Frames shared by servers and hosts

*Table 2.2-1: RHEV on RHS Hardware Information*



## 2.3 Software

The following table includes version information of components used in testing.

<b>Servers</b>	kernel-2.6.32-220.23.1 (RHEL 6.2z) glusterfs-3.3.0.7rhs-1
<b>Hosts</b>	kernel-2.6.32-279.22.1 (RHEL 6.3 + RHEV 3.1 host RPMs) glusterfs-3.3.0.7rhs-1, glusterfs-fuse-3.3.0.7rhs-1(update 4) vdsm-4.9.6-44.0.el6_3 KVM: cache=none, io=threads, format=qcow2, with six backing images to avoid all images residing on any one replication pair)
<b>Virtual Machines</b>	kernel-2.6.32-279.24.1 glusterfs-3.3.0.7rhs-1
<b>RHEV Manager</b>	kernel-2.6.32-279 RHEV-M: v3.1.0-22 vdsm-bootstrap-4.9.6.39.0.el6_3

***Table 2.3-1: RHEV on RHS Version Information***



## 3 Workloads

A set of non-overlapping, pure workloads representing the extremes of I/O activity found in typical applications were applied to this system. The alternative is to attempt to apply a variety of mixed workloads but either there are too many mixed workloads to consider or else it is risky and difficult to identify which mixed workload best represents a given application.

- *large-file* workloads - the iotest benchmark (<http://www.iozone.org>) was used (with **-+m** option to distribute processes across VMs).
  - sequential (32 GB per server total)
  - random (4 GB per server total)
  - reads and writes (64 KB records)
- *small-file* workloads – the smallfile benchmark was used (with **-host-set** option to distribute threads across VMs).
  - *create* – open new file, write 4 KB, close file
  - *append* – open existing file, write 4 KB, close file
  - *read* – open existing file, read 4 KB, close file
    - *uncached* – drop cache on all VMs and RHS servers before read
    - *cached* – perform uncached read prior to another read
  - *rename* – rename file from one directory to another
  - *delete* – unlink file

A single thread was executed within each VM.

The configurations for testing varied in three ways

- the number of RHS servers (2, 4, or 8)
- the number of RHEV hosts (2, 4, 8 or 16)
- the VM I/O mount type (virtio-block or virtio-net)

resulting in a matrix of 24 configurations per workload. Two I/O access configuration options<sup>1</sup> utilizing Gluster for file access from within a VM were configured:

- **virtio-net** – The VM mounts the Gluster volume to access files within so data travels through the virtio-net driver.
- **virtio-block** – The VM accesses files in a local file system (an ext4 file system mounted on an LVM volume using **/dev/vda**) where the OS issues I/O requests via the virtio-block driver. The host's VM process (qemu-kvm) relays the I/O requests to a virtual disk image file on the Gluster volume.

---

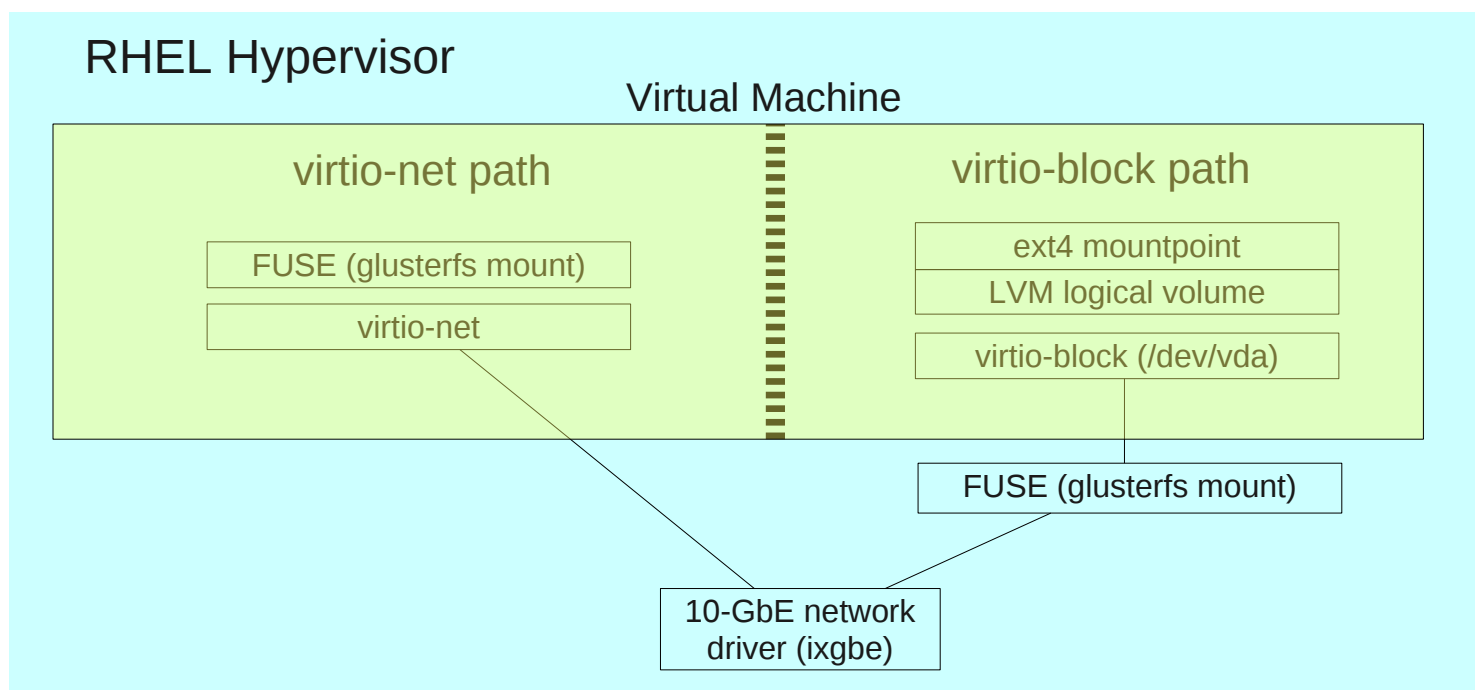
1 There are others including having the VM mount the Gluster volume via NFS but NFS has been extensively characterized in other contexts and the Gluster specific data paths were judged the most comparison relevant at this time.



The following figure illustrates the two I/O paths available to each VM. There is a wide variety of terminology applied to virtualization and cloud, so it seems worth the time to clarify this.

The *virtio-block* path identified below is also referred to as a “Live VM store”. In other words, the VM image is allowed to change and is persistent. The VMs (“guests” in KVM terminology), are actually backed by qcow2 images with a common backing image. This allows us to create a “golden image” VM in RHEV, provision it with all the software and configuration that it needs to function correctly, and then make a RHEV template from it, which can be used to clone this golden image into a set of guests sharing the invariant parts of the golden image.

The *virtio-net* path identified below is also referred to as a “virtual disk store for guests”. We create the guests in the same way that we did with the virtio-block path, but we do not run the workload inside the guest's disk image. Instead we create a Gluster mount point inside the guest and let the guest access the Gluster volume directly over the network.



**Figure 3-1: Virtio Paths Within the VM**





## 4 Results

Performance results generally consist of two basic categories: throughput and response time. Before running throughput tests, there was an effort to understand response time and its impact on the end-user of VMs running on virtualized storage.

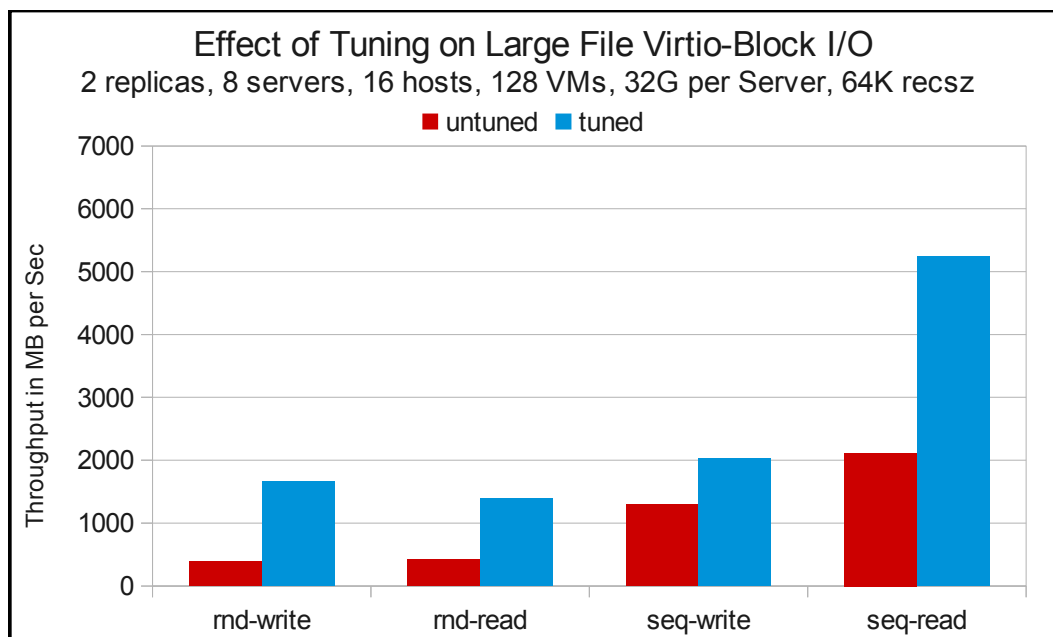
### 4.1 Tuning

Performance tuning has a significant impact on this system as the following graphs highlight. They measure the effect of tuning (excluding network and storage bricks which were measured elsewhere). Tuning changes reflected in these graphs include:

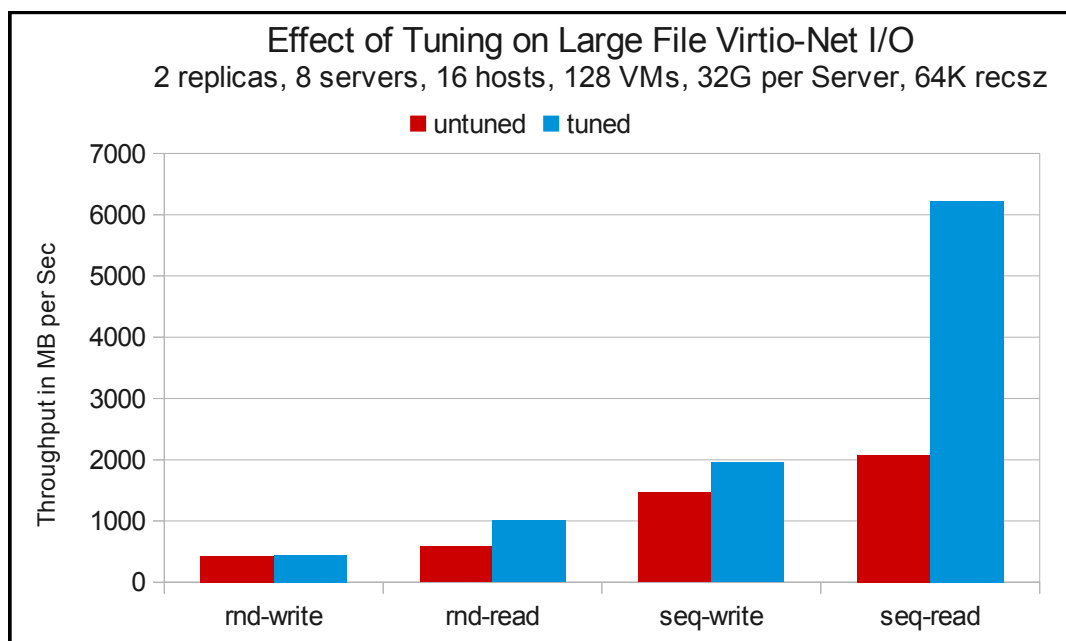
- **Kernel:** The *tuned* RPM package, distributed with RHEL6, supplies an assortment of predefined tuning profiles that configure a variety of kernel and block device parameters persistently across reboots with a single command. For example, the Gluster servers used the tuned RHS virtualization profile *rhs-virtualization*. This profile mounts the XFS brick with the *nobarrier* option as write barriers are not required because the PERC H700 MegaRAID has a non-volatile battery-backed write cache and disk write caching is disabled. Additionally, the profile forces use of the deadline I/O scheduler.
- **Gluster volume:** (applied using `gluster volume set <volume> group virt`):
  - disable quick-read, read-ahead, io-cache translators
  - enable eager-lock, remote-dio volume parameters
- **Server:** decreased queue depth to 128, tuned profile *rhs-virtualization*
- **Host:** tuned profile *virtual-host*
- **VM:** decreased queue depth to 8, tuned profile *rhs-guest*. This profile is not included in the RHEL, RHEV or RHS distributions but is identical to the *virtual-guest* profile distributed with RHEL 6.3 with the addition of increasing block device **read\_ahead\_kb** by 16x.
- **Network:** SELinux and iptables were disabled. Jumbo Frames (MTU=9000) were enabled on the 10-GbE NIC for each server and host.



## Large File



**Figure 4.1-1: Tuning Results on Large File Virtio-Block**



**Figure 4.1-2: Tuning Results on Large File Virtio-Net**

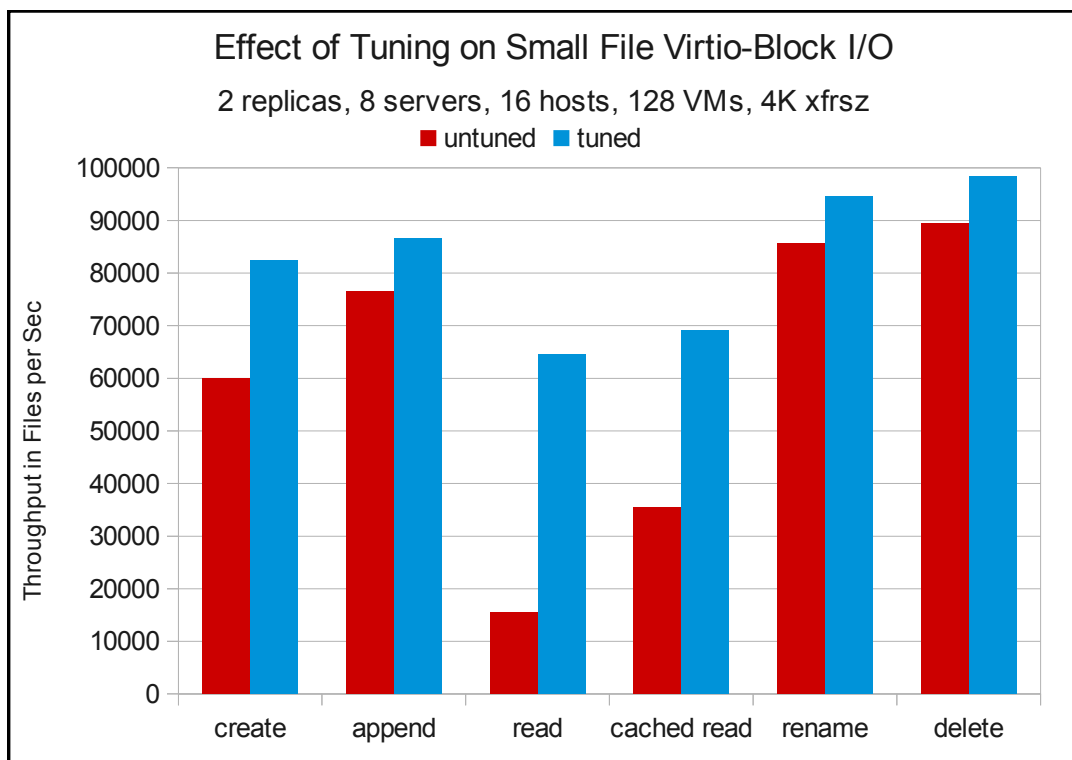
The impact of the tuning is clearly vital with regard to large file performance, particularly for:



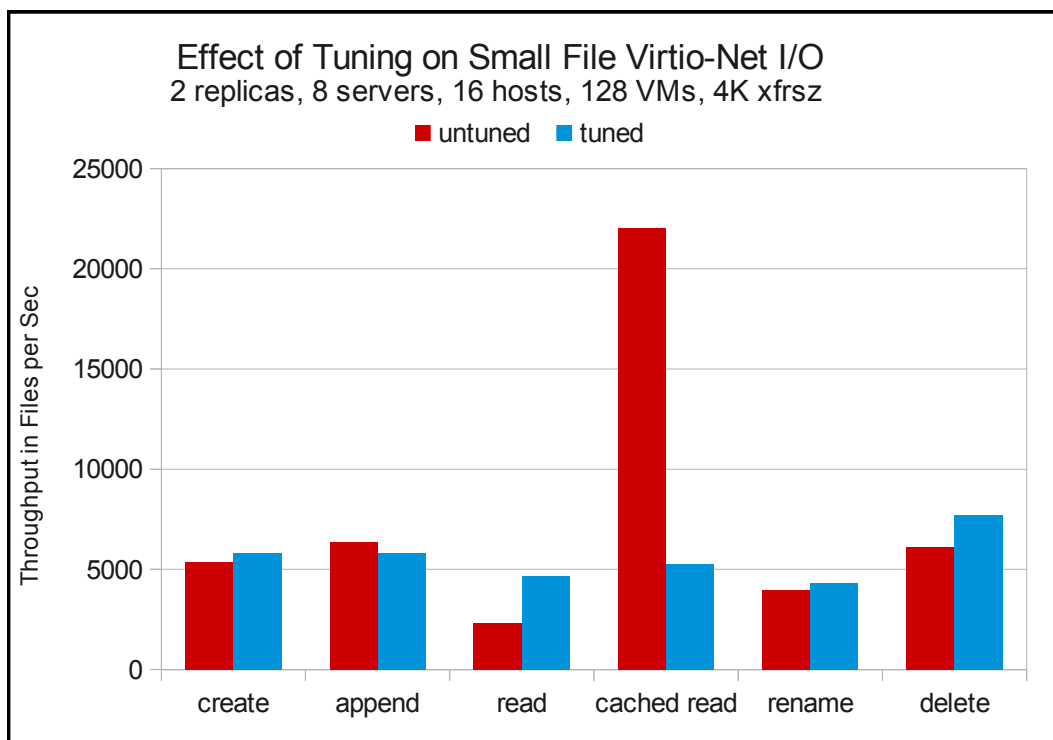
1. reads where the additional (adaptive) readahead in the block device and removal of (non-adaptive) readahead from the Gluster volume lessen the impact of I/O contention between guests.
2. random writes where *eager-lock* significantly reduces protocol overhead and *network.remote-dio* allows the RHS server to buffer writes (such as NFS) for more optimal I/O scheduling.

## Small File

For small files, there is only one case (virtio-net cached reads) where the tuning appears to significantly lessen performance. Every other case indicates the tuning is either neutral or significantly improves performance.



**Figure 4.1-3: Results of Tuning on Small File Virtio-Block**



**Figure 4.1-4: Results of Tuning on Small File Virtio-Net**

The case of virtio-net cached reads is most likely negatively impacted by disabling the io-cache translator in the VMs, a result of using the same volume for both the VM disk images and application file I/O. Ideally, system administrators would opt to provision separate Gluster volumes for virtualization and application I/O so each could be tuned appropriately for their intended workloads.



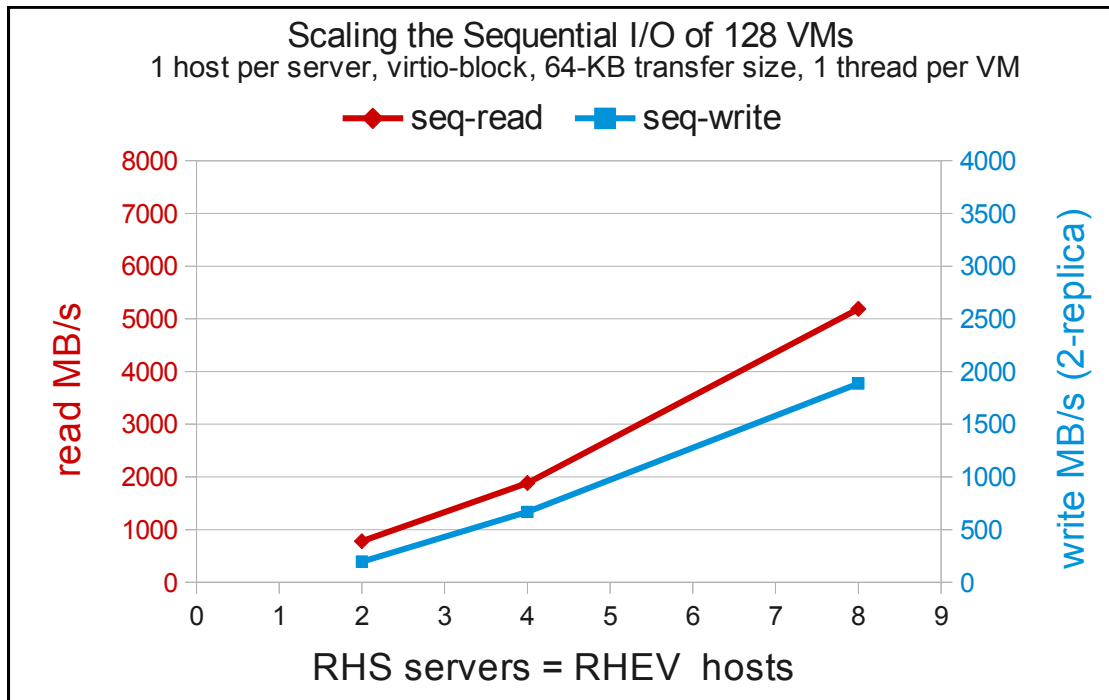
## 4.2 Scalability

Each of the scalability graphs present both read and write results with the read results measured on the left Y-axis while the write results are measured on the right Y-axis. Each axis is scaled so the top of the axis corresponds approximately to the maximum achievable result for the network configuration, in this case 1 10-GbE NIC per server and host. The X-axis is scaled starting at zero.

These graphs clearly indicate scaling but they also highlight that for this workload, placing so many guests into a single pair of RHEV hosts and RHS servers would produce less than optimal results. With 128 guests performing sequential I/O on two RAID LUNs, the workload is transformed into a somewhat random, non-sequential workload at the disk drives. Furthermore, additional context switching is introduced in the RHEV hosts.

### Sequential I/O

This figure graphs the performance of large file virtio-block sequential I/O as additional servers and hosts are added.

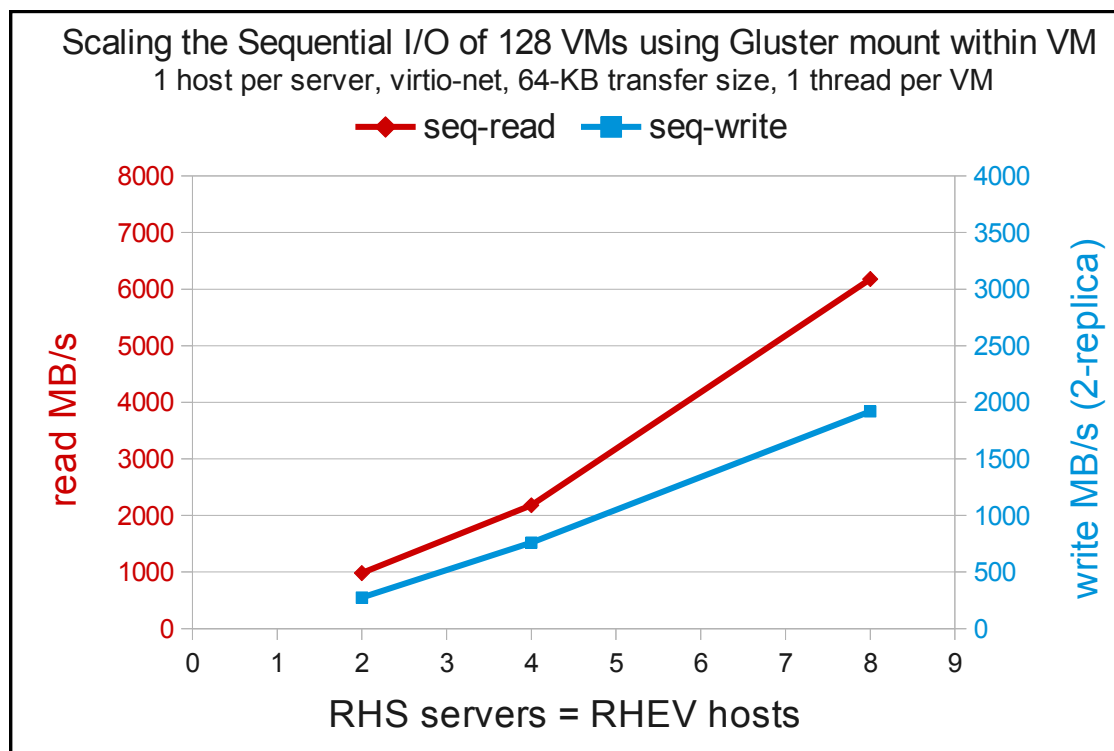


**Figure 4.2-1: Sequential I/O Performance via Virtio-Block**

Note that scaling is greater than linear but by the time scaling reaches eight servers and hosts, a significant percentage of the available network bandwidth (60% for reads, 45% for writes) has been achieved.



This figure graphs the performance of large file virtio-net sequential I/O as additional servers and hosts are added.

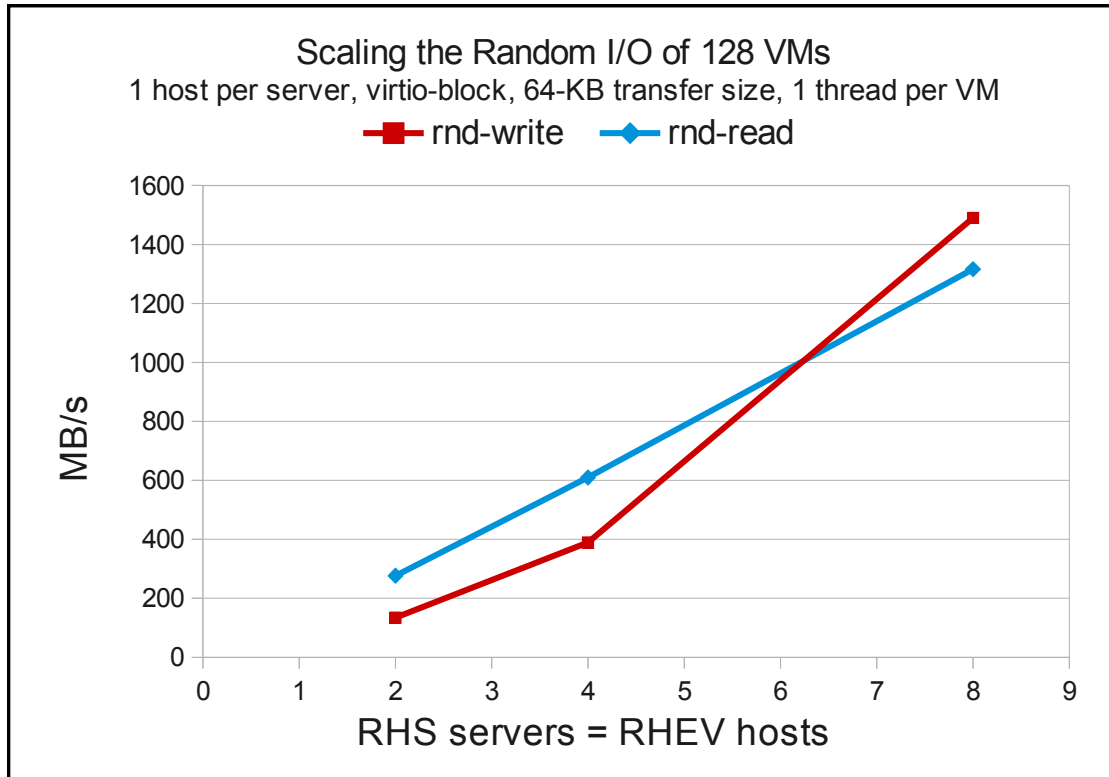


**Figure 4.2-2: Sequential I/O Performance via Virtio-Net**

Again the throughput increases with greater than linear scaling and the virtio-net path achieves roughly 80% of available network bandwidth on reads and approximately 50% on writes, slightly better than the virtio-block path throughput.

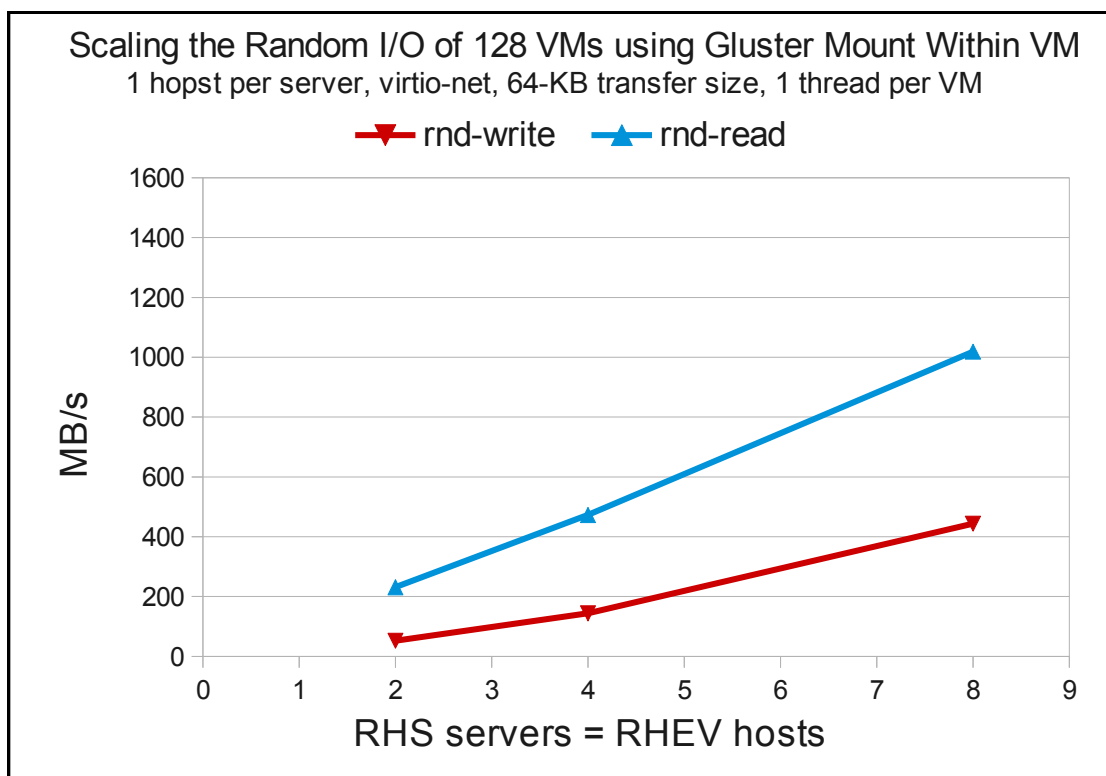


## Random I/O



**Figure 4.2-3: Random I/O Scaling Performance via Virtio-Block**

Given the hypothesis was that sequential I/O was being converted to random I/O when there are too many guests/server, the next two figures graph the throughput for a random I/O workload. Only one Y-axis is used in these graphs and it is not scaled to the maximum network bandwidth.



**Figure 4.2-4: Random I/O Scaling Performance via Virtio-Net**

The scaling for reads is almost perfectly linear, the strongest evidence of scalability thus far both in RHEV and RHS but the virtio-net path is approximately 25% slower. The most likely cause being Gluster client consumes significant CPU resources and requires more context switching (between the glusterfs process and the application) and each VM has only one vCPU.

For random writes, the virtio-block path is 3x faster than the virtio-net path. The write curves also increase faster than linear. Note that the VMs can buffer writes so they are able to re-order and even merge write requests to reduce the non-sequentiality of these requests. As such random writes appear more as sequential writes in this workload than is preferred. It would take a great amount of time to perform a random write test sufficiently large enough to prevent this re-ordering effect (33 GB per server of data was used in testing).





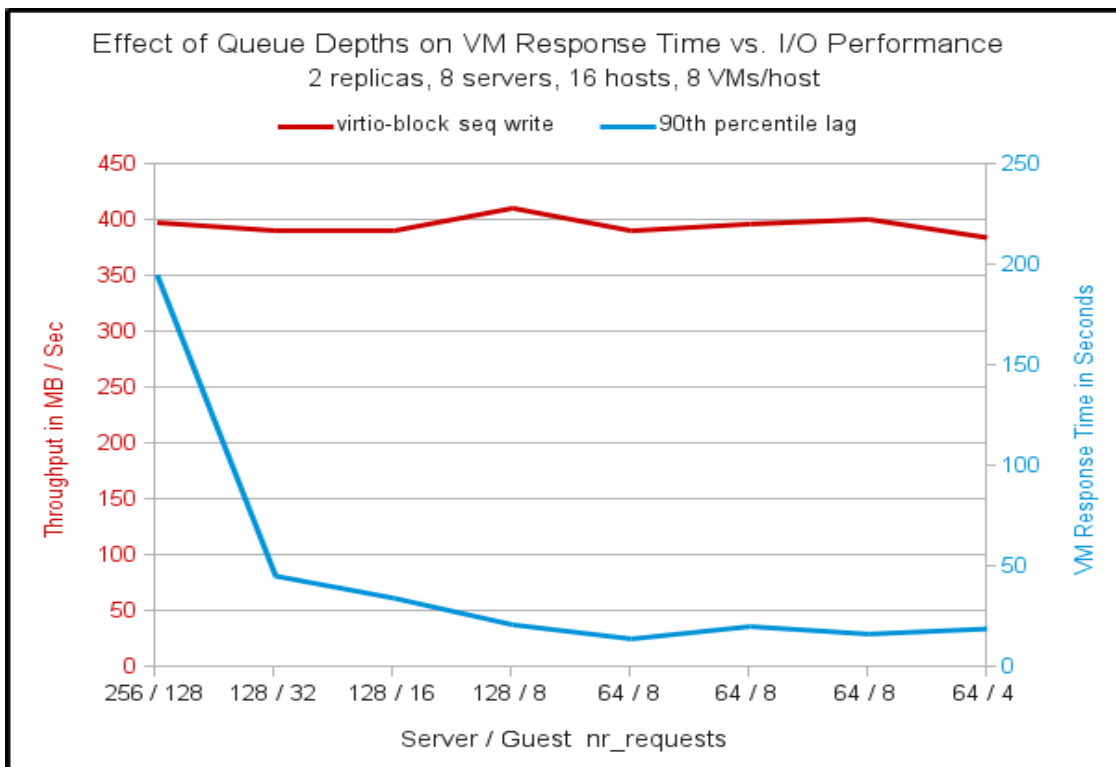
## VM Response Time

One concern with virtualization is the inevitable increase in response time that results from sharing physical hardware with other VMs. Prior to the lengthy procedure of executing the defined test matrix, testing was performed to determine if any system tuning would provide reasonable response time while achieving good throughput. While the higher throughput is desired, it is obviously not worth reducing the VMs to an unusable state.

Initial testing highlighted problems with response times for basic VM operations with 90% percentile response times measured in minutes in some cases, an unacceptable environment. It was observed that the response time of VMs could be greatly reduced without much loss of throughput by two changes:

- reducing the queue depths on the block devices in the VMs from 128 to 8 I/O requests
- reducing the block device queue depth on the Gluster servers from 256 to the Linux default of 128 I/O requests

The following figure graphs effect of queue depth tuning on both throughput (in MB/s) and VM response time of VMs for a large file, virtio-block, sequential write workload concurrently with a command to copy a small file locally in parallel to all 128 VMs via ssh (see Appendix C for command). The sequential write workload proved to be the worst case for inducing high response times in VMs.



**Figure 4.2-5: Analysis of Server/VM Queue Depths vs. Seq I/O**

Response times drop steadily until optimal tuning is achieved and does not drop further but the I/O throughput does begin to drop as queue depths are reduced. This is because VM block device queue depths are effectively added to the queue depth of the block device in the



RHS server. As such, with 8 VMs at 128 requests/VM the maximum queue depth is greater than 5 times larger than it would be for a bare metal server. A write workload is more able to fill these block device queues because the application does not have to block before it issues the next request. Consequently, sequential writes resulted in worst-case response times.

**Note:** not all workloads will benefit from such VM tuning. If there are few VMs with high I/O requirements, it is not recommended to reduce **nr\_requests**. However, with many VMs and concern about fairness and response time, this technique can be effective.

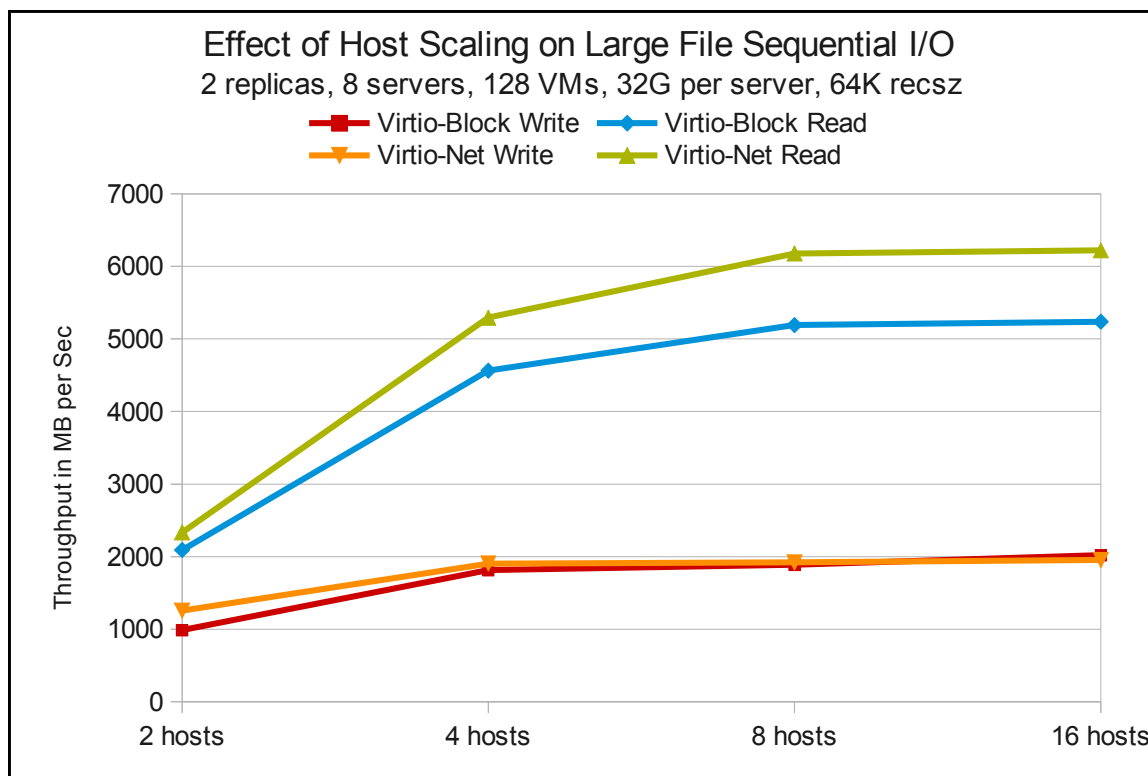
## 4.3 Scaling Hosts

To get a more high-level view of the results, graphs are presented indicating how throughput scales with RHEV hosts when eight RHS servers are used. Note that eight RHS servers with replication corresponds to four replication pairs of servers. For writes or metadata changes, both members of each replication pair must interact with the Gluster client. With reads however, Gluster need only access one replica on a single server.

Recall that each host has 24 cores so the VM:core ratio does not exceed one until eight hosts are used.

### 4.3.1 Large File I/O

This section examines large file sequential I/O throughput for a fixed storage server count.



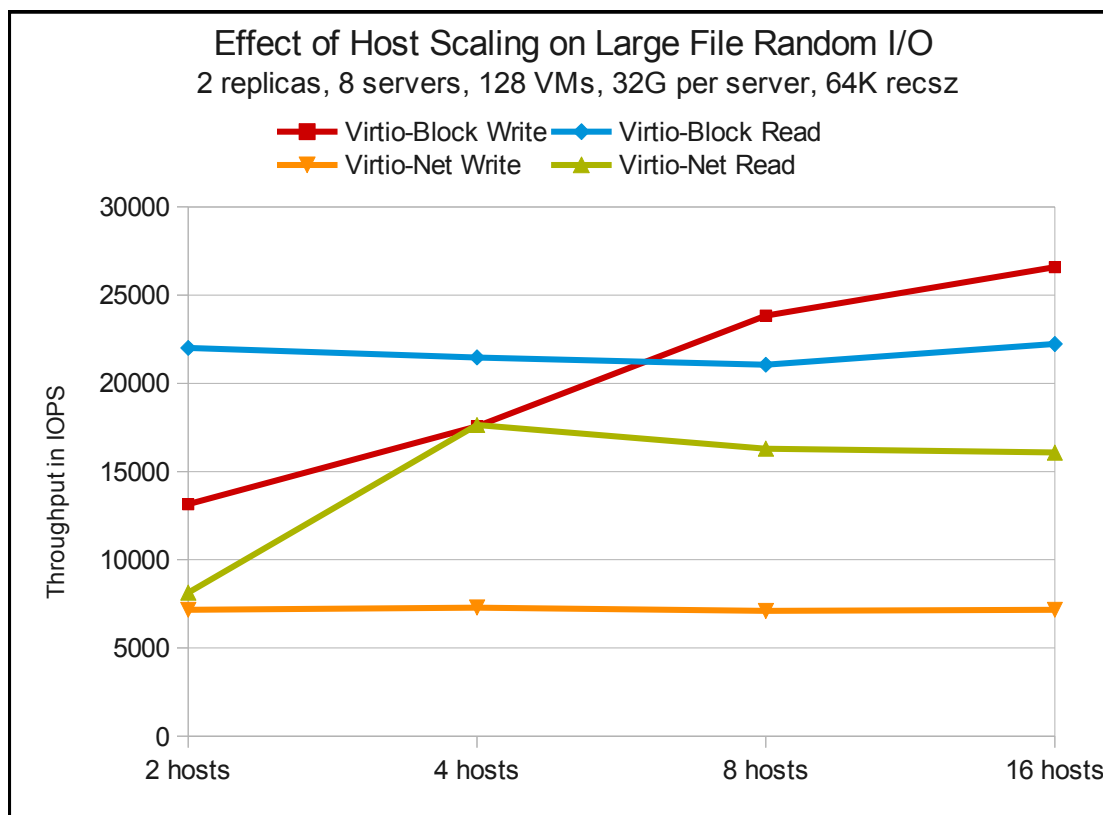
**Figure 4.3.1-1: Scaling Hosts for Large File Sequential I/O**



Sequential read scaling is good up to four hosts and scaling ceases when eight hosts are used. This is expected behavior for a sequential read workload where the network is typically the bottleneck and 75% of network and storage speeds are achieved. The difference between virtio-net and virtio-block reflects the increased cost of running a Gluster mount per VM in the virtio-net case, whereas with virtio-block the cost of the gluster mount is amortized across a number of VMs and there is less context switching (only one Gluster mount instance/host).

Sequential write scaling ceases at four hosts because the Gluster client must generate twice the network traffic for writes and synchronously write to all 128 streams.

Note the difference in units between the sequential I/O graphed above and random I/O graphs. Gluster is most efficient at large file sequential I/O with large transfer sizes. As a result the virtio-block and virtio-net curves for each workload are quite close. With 128 VMs reading and writing sequentially, the throughput is respectable considering the amount of concurrent streams (16 streams/server reads, 32 streams/server writes).



**Figure 4.3.1-2: Scaling Hosts for Large File Random I/O**

Random reads reflect a significant advantage for virtio-block with two hosts while virtio-net only achieves 2/3 the throughput of virtio-block by the time four hosts are engaged. Random I/O places a greater load on Gluster because the writes cannot be aggregated. At two hosts there are 64 VMs/host and each VM has approximately half a physical core available to it on

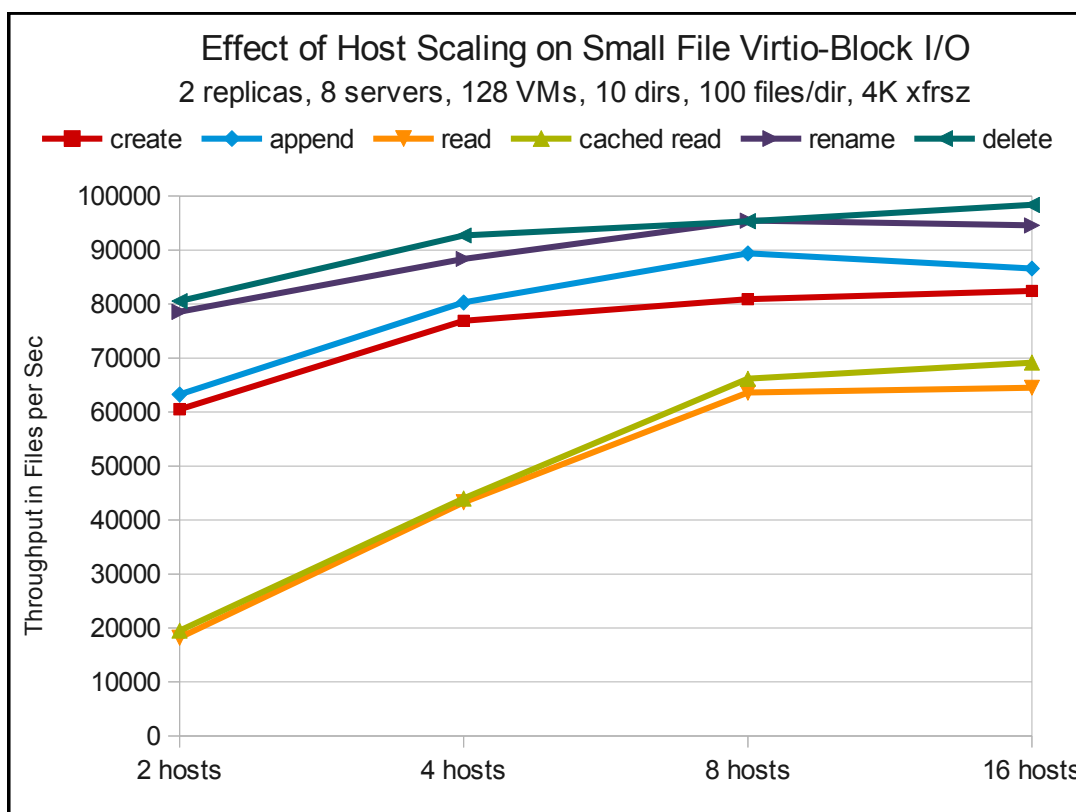


average. Each of these VMs must run its own Gluster client with all of its context switching. It would appear that Gluster's extra CPU load is responsible for the throughput increase from two to four hosts.

For random writes, virtio-block has the clear advantage because the VMs can buffer the writes while virtio-net cannot.

### 4.3.2 Small File I/O

This section examines small file I/O throughput as a function of hosts with a fixed storage server count.



**Figure 4.3.2-1: Scaling Hosts for Small File via Virtio-Block**

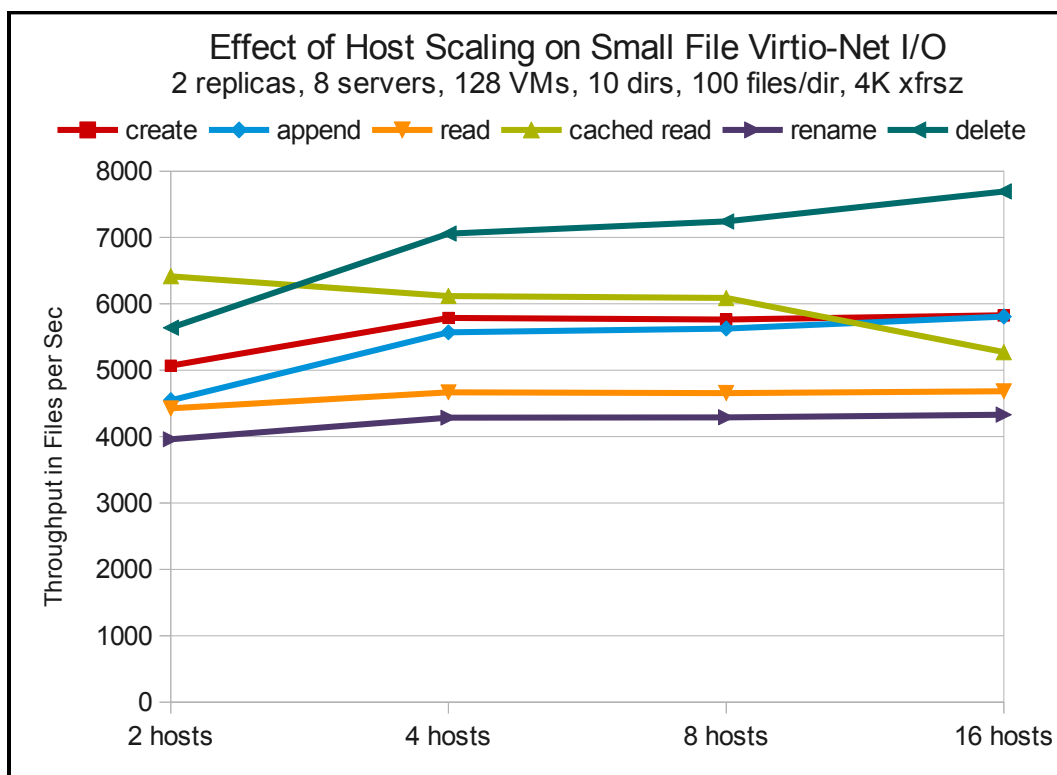
#### virtio-block

- Operation types involving modification of the file system do not scale with hosts and stop increasing at approximately four hosts, whereas read operations scale up to eight hosts.
- There is little difference between cached and uncached read operations. This is surprising until realizing the round trip behavior of these two cases is roughly the same because there is insufficient memory in each VM to buffer all of the 4-KB files that the VM would read, but the metadata (directories, etc) would be cacheable. For example, each VM reads 32K files for a total of 128 MB of data but the VM itself only has 512 MB to run the Linux OS.



- Reads start out at the 2-host level with much lower throughput, scale perfectly up to four hosts, and level off by the time eight hosts are engaged. This suggests that processing in the server is the bottleneck by the time eight hosts are involved. In fact this is observed with host read throughput reaching 600 MB/s in one case. However, the other host in the replica pair was idle during this test. This is an opportunity for RHS enhancement in balancing the load evenly across both members of replication pair.
- Note there is little difference between the **create** and **append** graphs. Although there is a large difference in the behavior of create and append operations at the brick layer, there is little difference at the Gluster protocol layer. As such, the bottleneck is at the protocol processing level and not at the disk I/O level. Even metadata updates such as those caused by the **rename()** and **unlink()** system calls appear this way. The number of round trips to the RHS servers is not much different in each case.

Note the factor of 10 difference in the Y-axis in the graph for virtio-net. This is primarily due to the fact that each VM must support overhead of a Gluster mount with its separate FUSE process, as well as extra round trips are required between the Gluster client and the host to access individual small files inside the Gluster volume. Whereas in the case of virtio-block, the Gluster volume is unaware of the small files being accessed in the test and the workload appears as random I/O to the virtual machine disk image.



**Figure 4.3.2-2: Scaling Hosts for Small File via Virtio-Net**

### virtio-net

The round-trip behavior of Gluster mounts in the VM is very different. The most scalable operation is **delete** while the **read** and **cached-read** operations start out as expected with



**cached-read** being faster as metadata does not have to be retrieved. However, by the time 16 hosts are engaged, **cached-read** performance equals that of (uncached) **read**. This may be surprising but recall that the total amount of memory available for caching is in the eight RHS servers (Gluster client does not cache).

## 4.4 Detailed Performance

### 4.4.1 Large File I/O

Results are presented as pairs of surface plots.

- Y-axis (vertical) indicates throughput
- X-axis (left to right) indicates number of hosts
- Z-axis (front to back) indicates number of RHS servers

Both X-axis and Z-axis are logarithmic-scale graphs because powers of two were used as data points to minimize the number of points to collect over a broad range. This results in some odd appearance when throughput is scaling linearly with two, four and eight RHS servers.

Graphs are presented in pairs with virtio-block and virtio-net graphs side by side in order to compare the behavior of virtualized storage when accessed via these different paths. When the Y-axis of the two graphs *does not have the same scale*, the virtio-net graph is shortened to call attention to it. In all graphs, throughput is expected to rise as the data progresses to the right (increased hosts) or move to the back of the graph (increased servers). The object is to determine what ratios (hosts:server and VMs:host) results in best performance.

The **iozone** (<http://www.iozone.org/>) benchmark was used for large file I/O with the “**-+m**” option that enables distributed workload generation. Workload categories are

1. **read vs. write** – direction of data transfer
2. **random vs. sequential** – sequential I/O transfer represents file access starting with first byte of file in order of byte offset until last byte of file is reached. Random I/O is the opposite where chunks of the file are transferred in random order, without repetition, until entire file is transferred.



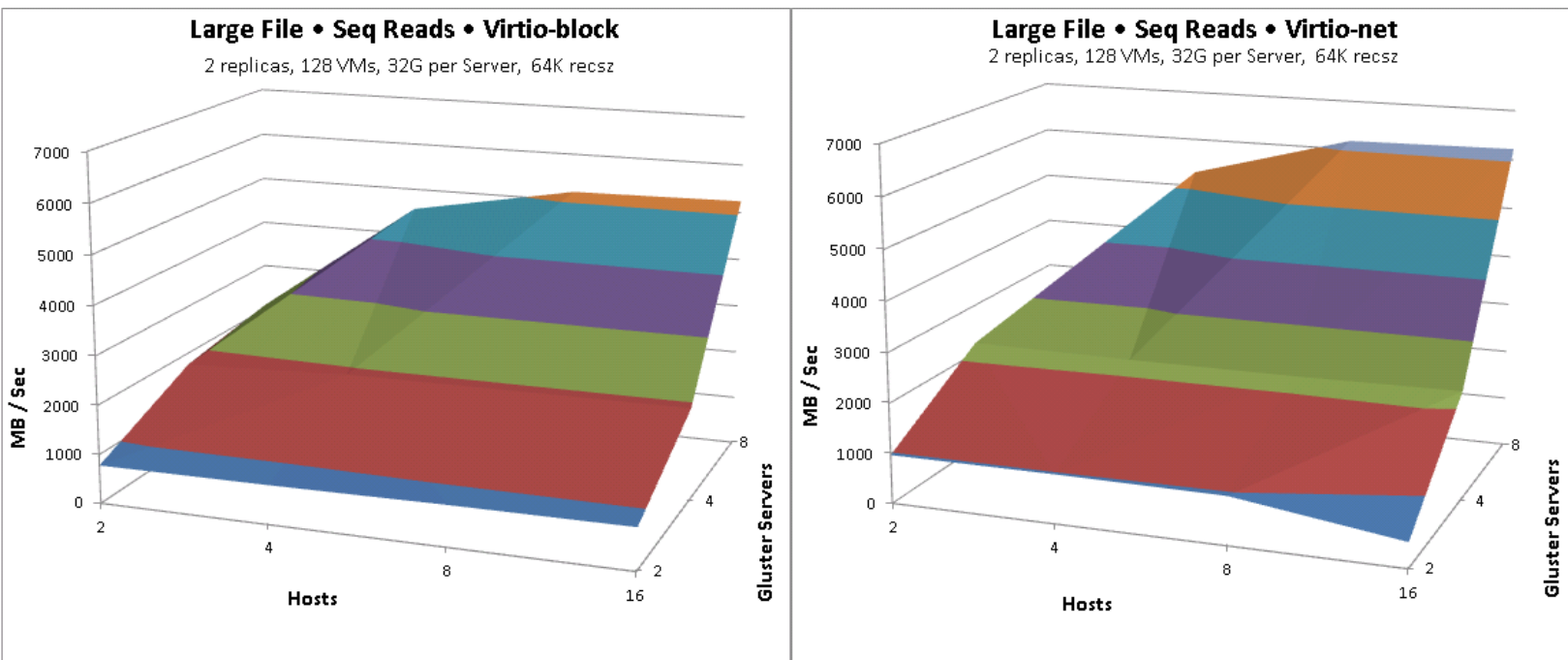
For example, the following command will initiate I/O using a separate thread on each of two VMs on a single host writing 8 GB/thread in 4-KB transfer sizes with no rewrite test:

```
iozone -+m 2vm.ioz -w -c -e -i 0 -+n -r 4k -s 8g -t 2
```

where 2vm.ioz contains records such as:

```
gprfvm-0003-10ge /mnt/test /usr/local/bin/iozone
gprfvm-0007-10ge /mnt/test /usr/local/bin/iozone
```

## Sequential Reads

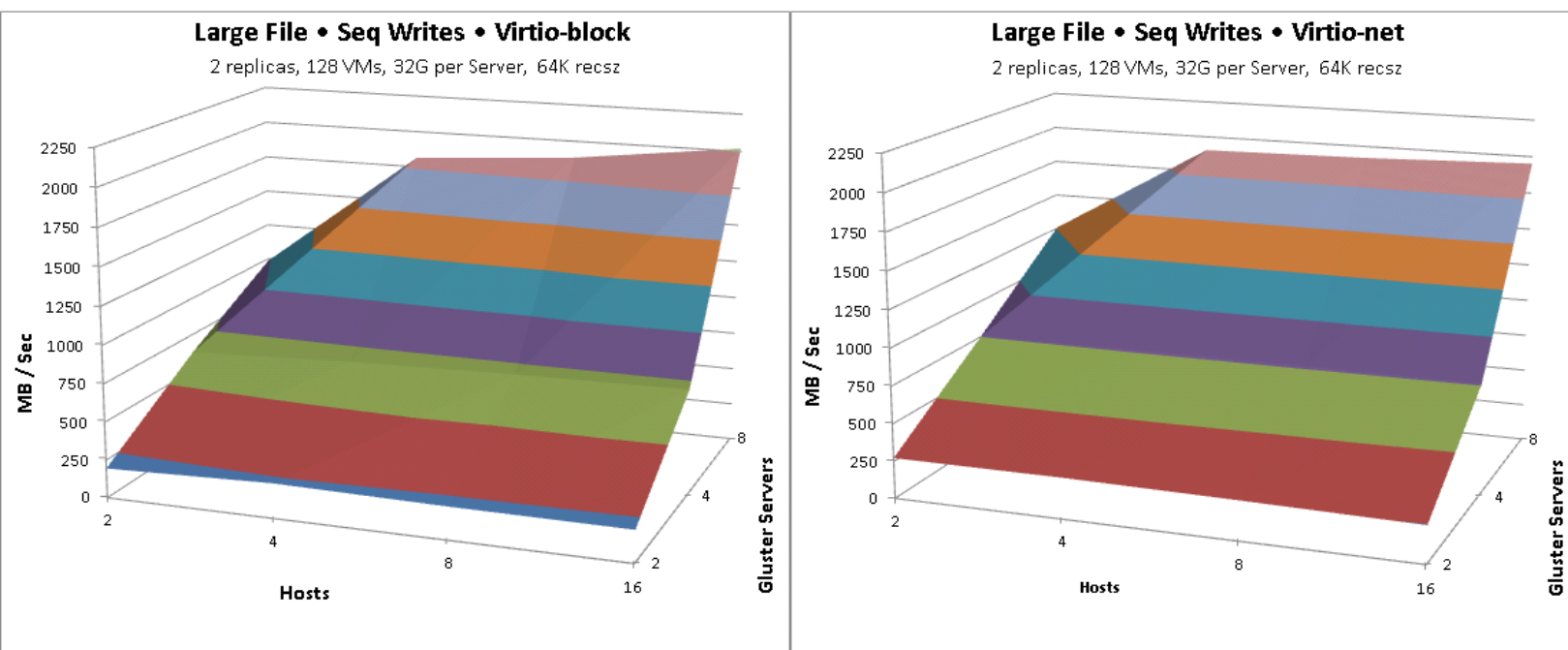


**Figure 4.4.1-1: Sequential Read Results**

In the sequential read workload, similar results are observed with either the virtio-block or virtio-net path, with best results at eight hosts and eight servers. Note that at two Gluster servers, the throughput does not vary with the number of hosts at all. This is entirely reasonable for sequential reads which is typically a network-limited workload. With eight hosts and eight servers, 6100 MB/s is achieved, slightly less than 800 MB/s/server, a respectable result for 128 VMs reading in parallel. This result is achieved because of XFS prefetching configured by the kernel tuning.



## Sequential Writes



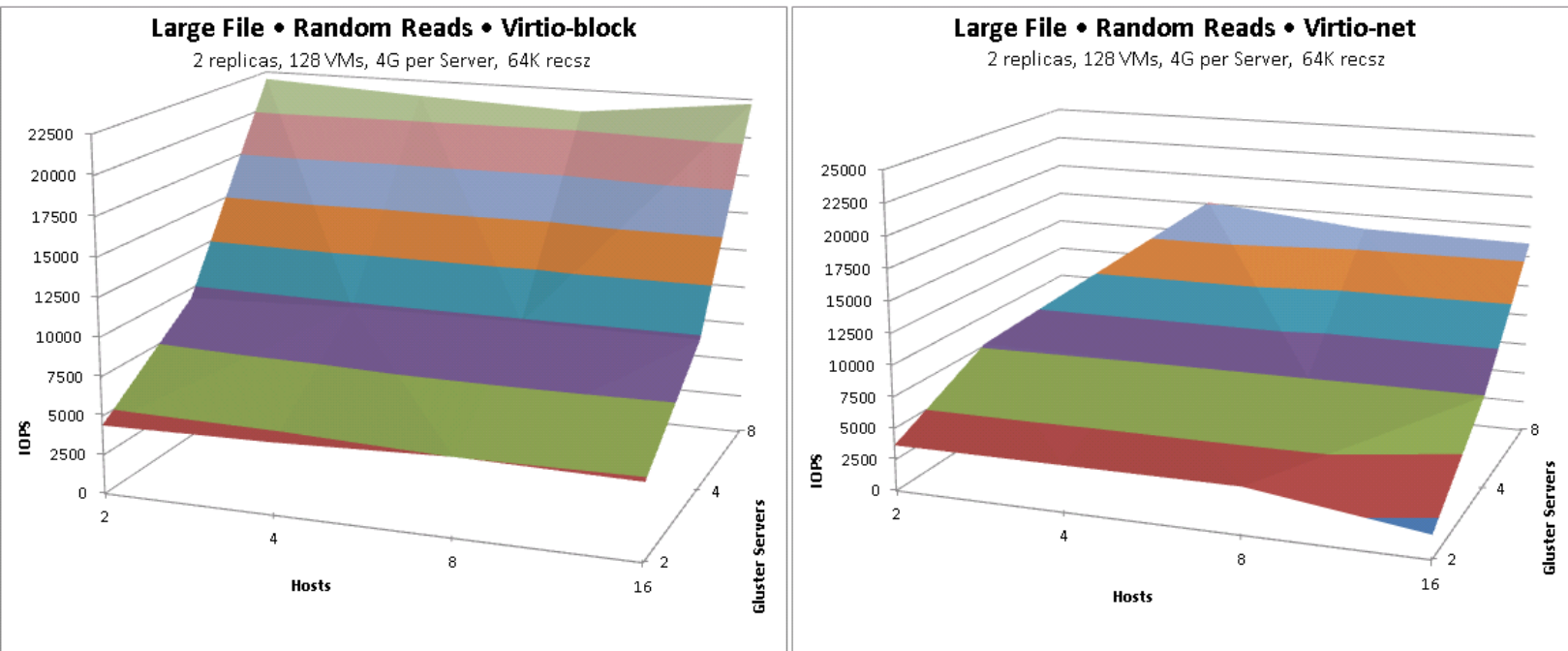
**Figure 4.4.1-2: Sequential Write Results**

In the sequential write workload, with two RHS servers throughput does not vary with host count at all. The virtio-block graph shows best performance with eight servers and 16 hosts. Whereas with virtio-net, the peak throughput appears to occur at four hosts and eight servers. Peak throughput is approximately 2000 MB/s, or 250 MB/s/server. With a replication factor of 2x on network, the maximum network limited throughput would be roughly 4000 MB/s, or 500 MB/s/server. Write throughput is somewhat limited by the requirement to synchronize replication and resulting latency.





## Random Reads

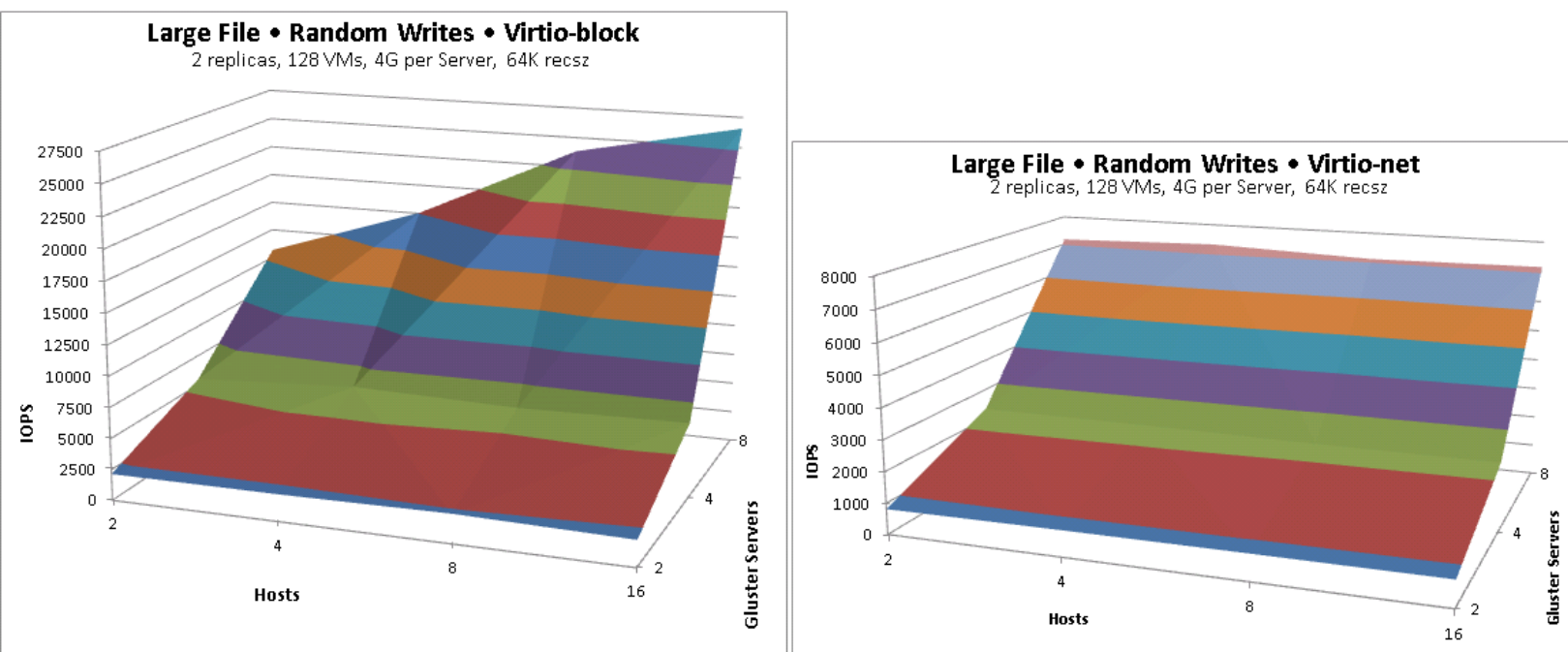


**Figure 4.4.1-3: Random Read Results**

For random reads, the virtio-net path appears to offer significantly lower throughput and the throughput is relatively insensitive to the number of hosts. This workload is typically limited by how fast disks are able to seek. Thus the number of Gluster servers should determine random throughput. However, the throughput is expected to be the same whether doing virtio-block or virtio-net I/O. This suggests that there is additional overhead in the client I/O path when virtio-net is running. One factor is that there are many more Gluster clients (128) interacting with servers in this case. Another factor is the additional caching and prefetching occurring in virtio-block tests. Since the amount of data (4 GB/server) was far less than the amount of memory, the virtio-block configuration would eventually be able to contain the entire data set within memory, whereas in the virtio-net case there would be no caching in the VM or host. Note the dip at 16 RHEV hosts and two RHS servers. This is the point at which the RHS server is being pushed into I/O contention. In fact, with virtio-net case iostat data indicates that from the server that the maximum I/O queue depth was achieved with this workload (128 requests), further indicating server I/O saturation at 700 reads/sec and 150 writes/sec.



## Random Writes



**Figure 4.4.1-4: Random Write Results**

For random writes, there is a factor of 5x drop from virtio-block peak performance to the virtio-net peak. Recall that the same amount of data must travel over the network. The difference is because in virtio-net, the Gluster client in the VM does not buffer writes and there is no I/O scheduling. Thus all the work is performed in the Gluster server. This is reflected in the virtio-net graph where throughput is dependent of the number of servers. Whereas the kernel of the virtio-block VM is using its buffer cache and block device queue to schedule and aggregate writes before they reach Gluster. This can result in a larger average I/O size being issued to Gluster as well as a more optimized write sequence.



## 4.4.2 Small File I/O

The small file benchmark (<https://github.com/bengland2/smallfile>) is used to generate file operations. For example, to create 32K files of size 4KB in a Gluster mount point using 1 thread (process) in each of 8 VMs, you can use the command:

```
smallfile_cli.py -top /mnt/glusterfs/smf.d -host-set  
"g1,g2,...,g8" --threads 1 -file-size 4 -files 32768
```

For small files, the differences between virtio-block and virtio-net are much greater, because Gluster client is receiving a very different workload in the two cases. For virtio-block case, the small files are being created on an ext4 file system embedded within a large virtual disk file in the Gluster volume and as such Gluster is receiving a mixture of read and write requests to a set of large virtual disk image files (e.g., multiple inodes can be fetched by a single Gluster read of the region in the virtual disk image where the inode table is located). Additionally, in the virtio-block case metadata can be cached by the VM OS to eliminate round trips to Gluster servers. Whereas in the virtio-net case, each small file requires both metadata and data round-trip operations being sent to both Gluster servers containing the replicas for that one file. Thus there are far more round trips and synchronizations involved in a small file virtio-net operation.

### Creates

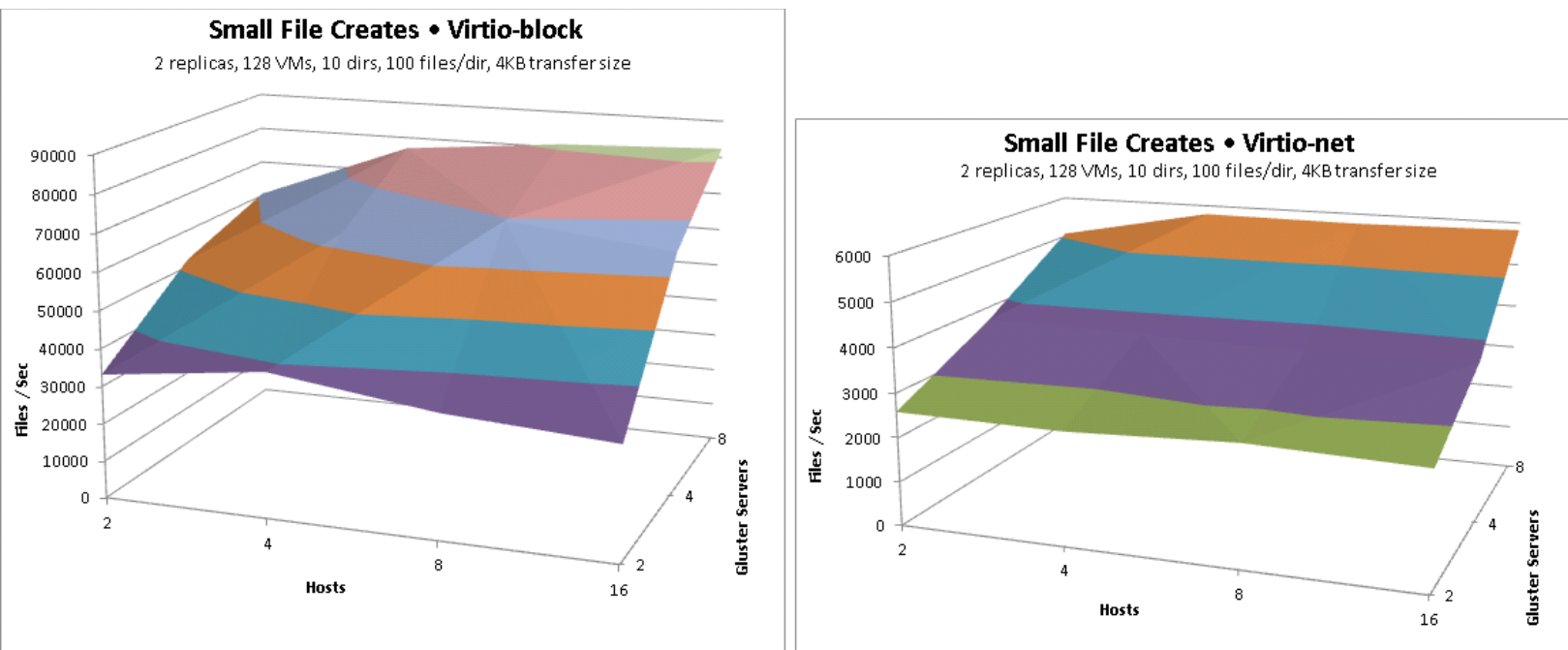
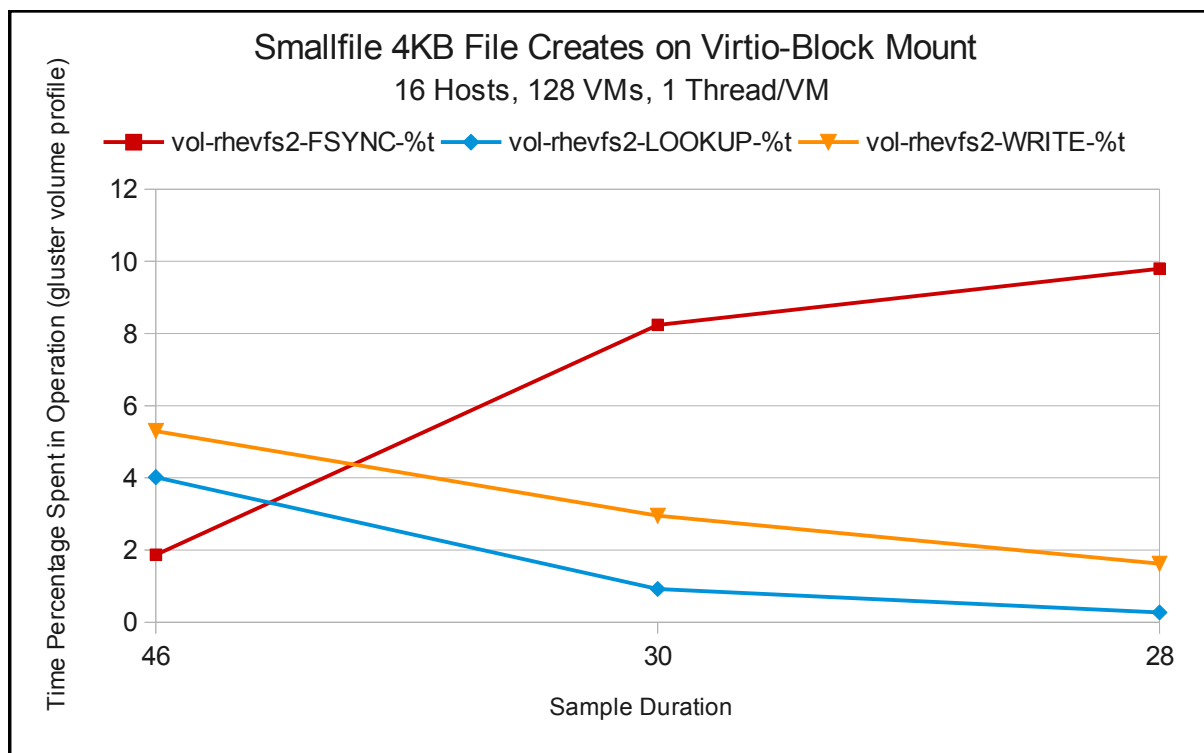


Figure 4.4.2-1: Create Results



For virtio-block, peak throughput comes with eight servers and eight hosts, whereas with virtio-net, peak throughput comes with 16 hosts and eight RHS servers. There is greater than a 10 factor decrease from virtio-block to virtio-net.

For the virtio-block case, where Gluster files are already created, there is a much higher percentage of data transfer (WRITE) operations. As a result not so much time is spent in metadata operations and some of the writes are bigger than 4-KB (e.g., 8 KB). The following graph shows the percentage of time spent in Gluster operation types for the virtio-block case (smaller test run).



**Figure 4.4.2-2: Percentage of Time Spent Per Operation Type**

The next figure is a Wireshark screenshot focuses on Gluster round trips between one replication pair of servers and one host. The LOOKUP and FSYNC traffic is noise caused by the test harness. Note that FSYNC is the most expensive operation in terms of time. On average they take 100 msec. The writes proceed to the disk image files in parallel without blocking on I/O completion. Most of the writes are 128-KB writes but they complete in 9 msec on average. These are very large considering the files being written by the application are 4-KB in size. Write aggregation is the reason that virtio-block outperforms virtio-net.



smf-cr-vioblk.tcpdump [Wireshark 1.8.3 (SVN Rev Unknown from unknown)]

Filter: glusterfs and (ip.addr==172.17.40.21 or ip.addr == 172) Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
11128	1.387381	172.17.40.22	172.17.10.1	GlusterFS	214	V330 FXATTROP Reply (Call In 1
11131	1.387438	172.17.10.1	172.17.40.22	GlusterFS	4278	V330 WRITE Call (Reply In 1
11132	1.387459	172.17.40.22	172.17.10.1	GlusterFS	306	V330 WRITE Reply (Call In 1
11133	1.387465	172.17.10.1	172.17.40.21	GlusterFS	4278	V330 WRITE Call (Reply In 1
11134	1.387467	172.17.40.21	172.17.10.1	GlusterFS	306	V330 WRITE Reply (Call In 1
11135	1.387472	172.17.10.1	172.17.40.22	GlusterFS	17962	V330 WRITE Call (Reply In 1
11137	1.387521	172.17.10.1	172.17.40.21	GlusterFS	17962	V330 WRITE Call (Reply In 1
11140	1.387570	172.17.40.21	172.17.10.1	GlusterFS	306	V330 WRITE Reply (Call In 1
11142	1.387584	172.17.40.22	172.17.10.1	GlusterFS	306	V330 WRITE Reply (Call In 1
11144	1.387628	172.17.40.22	172.17.10.1	GlusterFS	306	V330 WRITE Reply (Call In 1
11146	1.387692	172.17.40.21	172.17.10.1	GlusterFS	306	V330 WRITE Reply (Call In 1
11148	1.387745	172.17.10.1	172.17.40.22	GlusterFS	278	V330 FXATTROP Call (Reply I
11151	1.387779	172.17.10.1	172.17.40.22	GlusterFS	278	V330 FXATTROP Call (Reply I
11152	1.387801	172.17.10.1	172.17.40.21	GlusterFS	278	V330 FXATTROP Call (Reply I
11153	1.387818	172.17.10.1	172.17.40.22	GlusterFS	178	V330 FSYNC Call (Reply In 1
11154	1.387831	172.17.10.1	172.17.40.21	GlusterFS	178	V330 FSYNC Call (Reply In 1
11158	1.387868	172.17.40.22	172.17.10.1	GlusterFS	214	V330 FXATTROP Reply (Call I
11161	1.387919	172.17.40.21	172.17.10.1	GlusterFS	214	V330 FXATTROP Reply (Call I
11163	1.387973	172.17.40.22	172.17.10.1	GlusterFS	214	V330 FXATTROP Reply (Call I
11165	1.388025	172.17.10.1	172.17.40.22	GlusterFS	230	V330 FINODELK Call (Reply I
11166	1.388043	172.17.10.1	172.17.40.21	GlusterFS	230	V330 FINODELK Call (Reply I
11169	1.388092	172.17.40.21	172.17.10.1	GlusterFS	306	V330 FSYNC Reply (Call In 1
11170	1.388103	172.17.40.22	172.17.10.1	GlusterFS	106	V330 FINODELK Reply (Call I
11172	1.388136	172.17.40.21	172.17.10.1	GlusterFS	106	V330 FINODELK Reply (Call I
11173	1.388138	172.17.40.22	172.17.10.1	GlusterFS	306	V330 FSYNC Reply (Call In 1
11182	1.388348	172.17.10.1	172.17.40.22	GlusterFS	178	V330 FSYNC Call (Reply In 1
11184	1.388369	172.17.10.1	172.17.40.21	GlusterFS	178	V330 FSYNC Call (Reply In 1
11188	1.388445	172.17.40.22	172.17.10.1	GlusterFS	306	V330 FSYNC Reply (Call In 1
11189	1.388445	172.17.40.21	172.17.10.1	GlusterFS	306	V330 FSYNC Reply (Call In 1
11195	1.388597	172.17.10.1	172.17.40.22	GlusterFS	178	V330 FSYNC Call (Reply In 1

Frame 11133: 4278 bytes on wire (34224 bits), 4278 bytes captured (34224 bits)

Ethernet II, Src: Intel\_04:7d:e1 (90:e2:ba:04:7d:e1), Dst: Intel\_19:f1:9c (90:e2:ba:19:f1:9c)

Internet Protocol Version 4, Src: 172.17.10.1 (172.17.10.1), Dst: 172.17.40.21 (172.17.40.21)

Transmission Control Protocol, Src Port: 1011 (1011), Dst Port: 24009 (24009), Seq: 10301, Ack: 2061, Len: 4212

Remote Procedure Call, Type:Call XID:0x01cbb06e

GlusterFS

- [Program Version: 330]
- [GlusterFS: WRITE (13)]
- GFID: 5e28e149-bfcf-4897-ae72-7662e8813ee5
- File Descriptor: 8
- Offset: 29230047232
- Size: 0
- Flags: 0260004000, O\_NONBLOCK, O\_NDELAY, Unused
- Dict, contains 0 items
- Data (4096 bytes)

File: "/perf1/bengland/public/rhs/virt/... Profile: Default

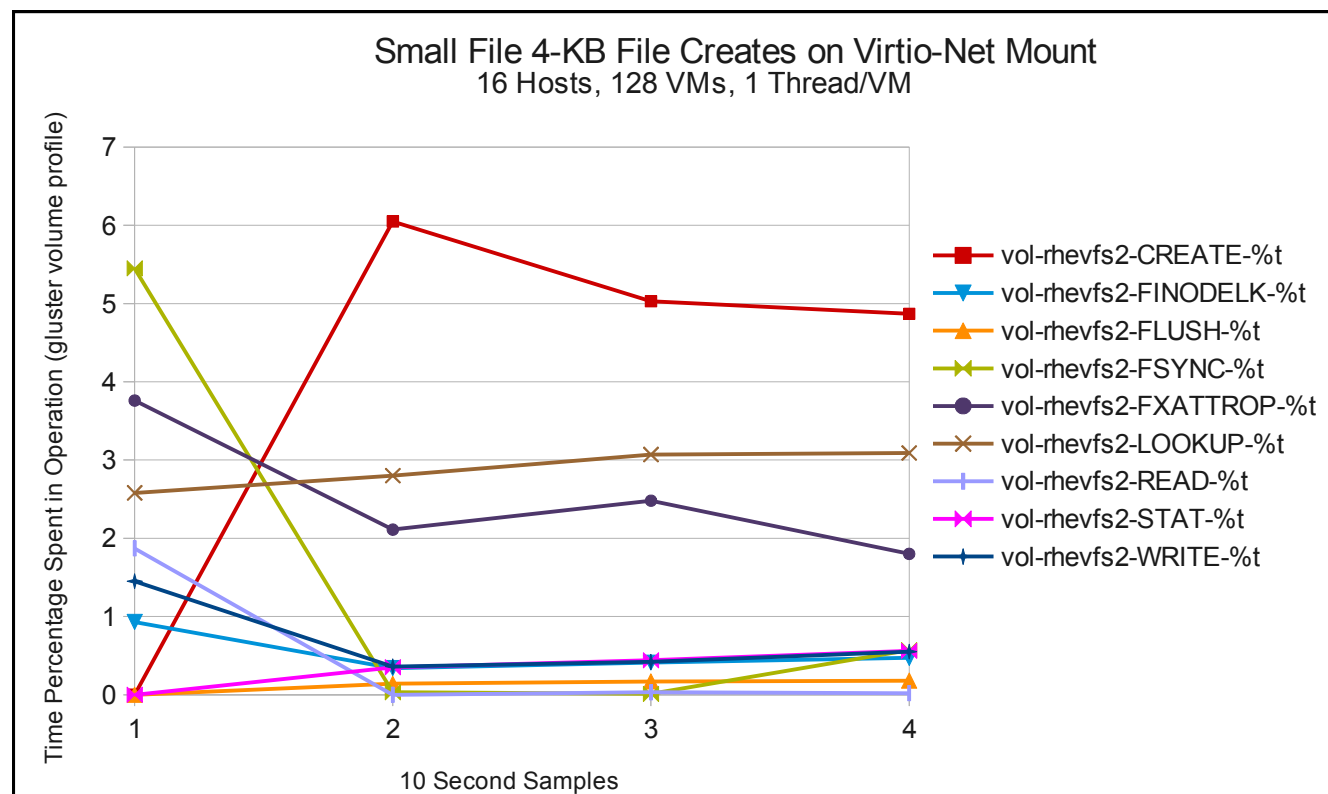
**Figure 4.4.2-3: Wireshark Data Capture**

In the virtio-net case, the VM is responsible for talking to the Gluster servers to create the file. There are many protocol round-trips that the VM must execute to create this 4-KB file, as shown in the wireshark Gluster plugin capture, such as the LOOKUP calls to verify the file already exists. These LOOKUP calls are performed on every brick in parallel for every file because the file could be located on a different brick than the one specified by the consistent hashing algorithm.





This figure graphs the percentage of time spent on Gluster operations during a small file virtio-net test. Note that the FXATTROP, LOOKUP and CREATE operations consume the majority of the time as opposed to the actual WRITE operations. Although not ideal, it is the logical result of having VMs as Gluster clients.

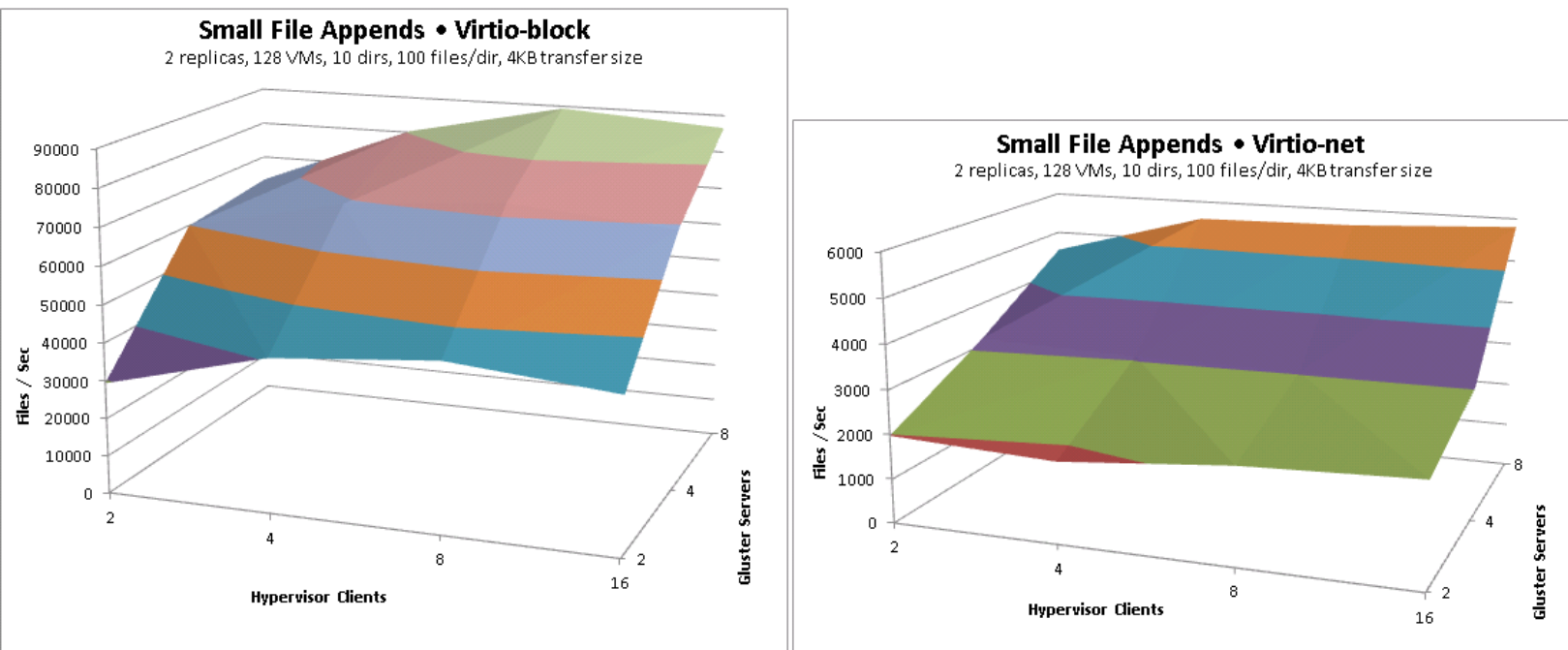


**Figure 4.4.2-4: Percentage of Time Spent in Gluster Operations**



## Appends

For small file appends, the benchmark is just appending data to an existing file. Although this may be expected to be significantly faster, it is not. In either case the bottleneck is not the metadata journaling associated with a new file. With virtio-block, there is scaling up through eight hosts (most likely due to buffering in VM) but virtio-net throughput is level as the host count increases indicating a bottleneck in the server count.

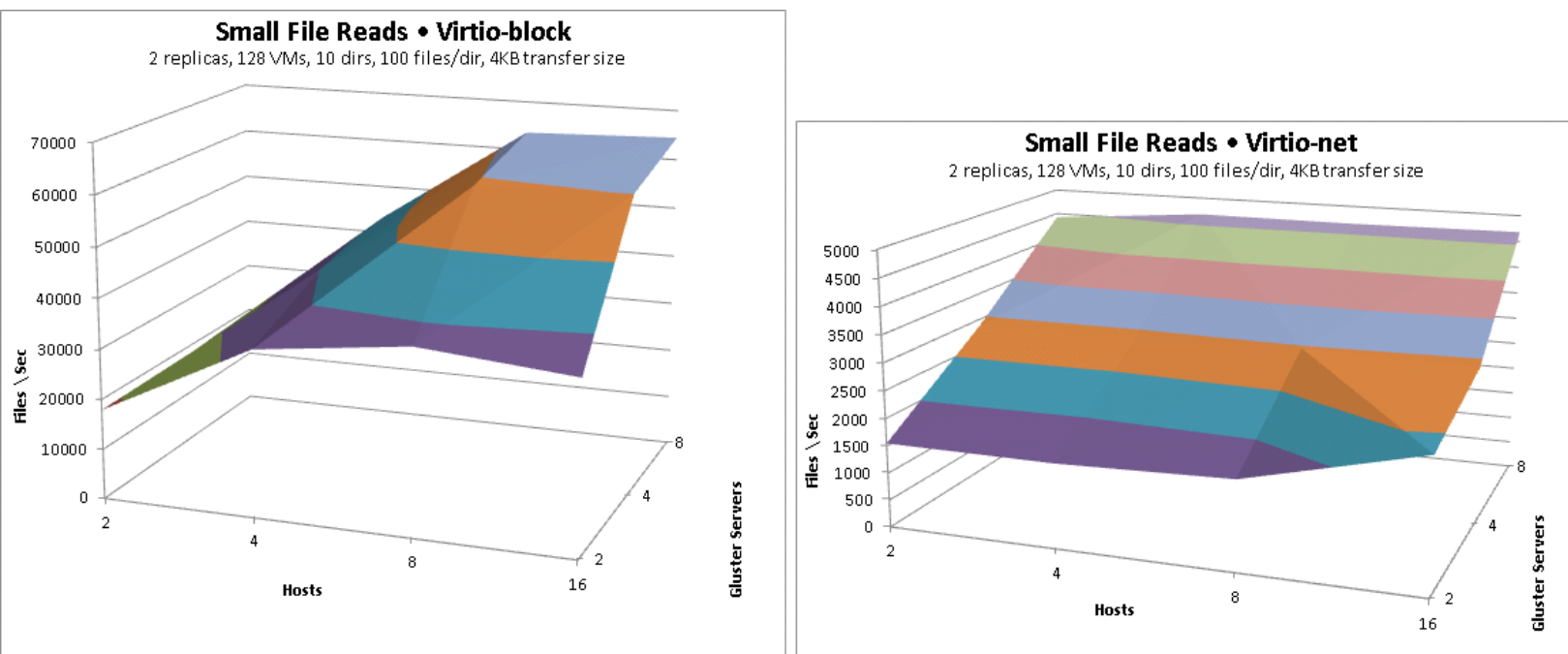


**Figure 4.4.2-5: Append Results**



## Reads

In the small file reads, cache is dropped on all VMs and servers prior to testing so data must travel from disk to application.



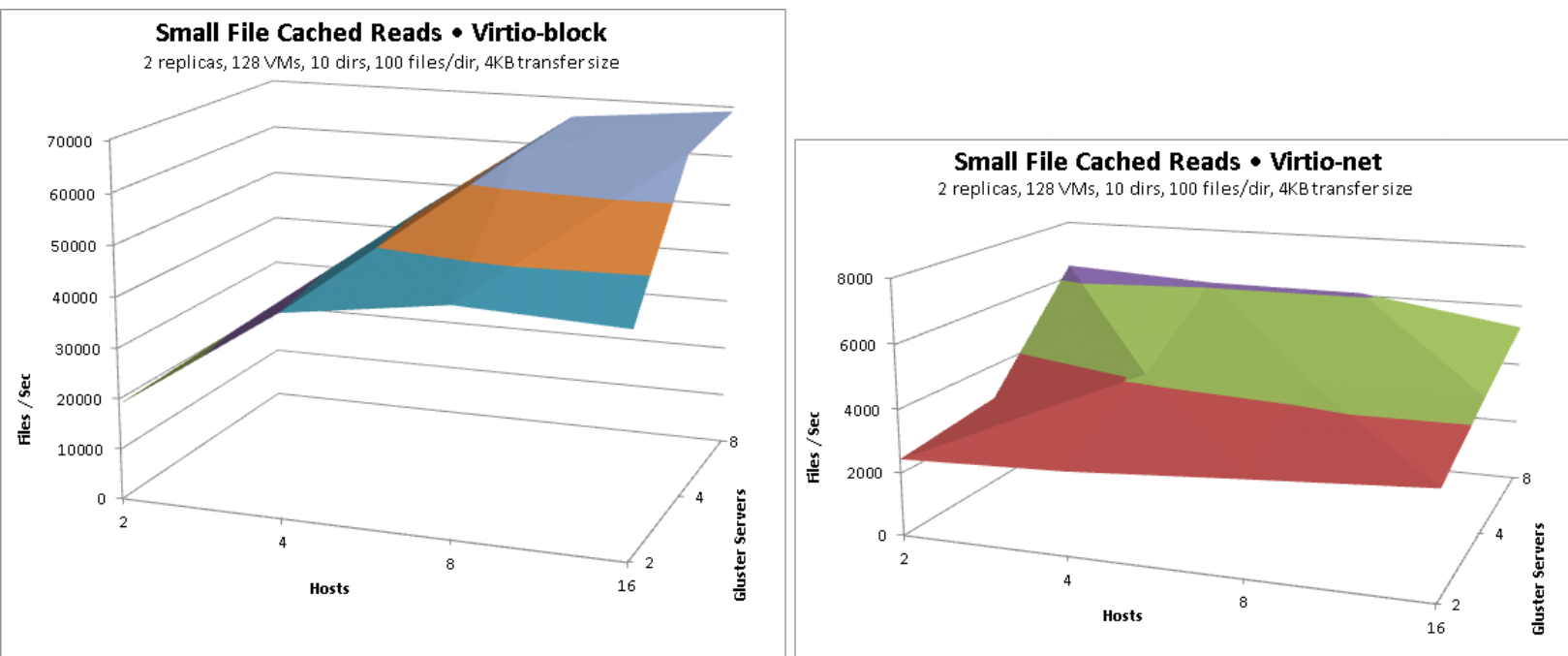
**Figure 4.4.2-6: Read Results**

The uncached read graphs above have radically different shapes for virtio-block and virtio-net. In the virtio-block case, almost half the maximum throughput is achieved with just four hosts and two servers, and close to all of the maximum throughput is achieved with eight hosts and four servers. Whereas for virtio-net, throughput is relatively insensitive to the number of hosts and is more directly related to the number of Gluster servers. The hypothesis on for virtio-block case is that VMs may have prefetched and cache much metadata whereas in the virtio-net case, there was no significant caching of metadata in the VM (Gluster client will not cache metadata for more than one second). This shifts the bottleneck from round trips to the server and disk accesses to metadata caching in the VM, where additional hosts may have resulted in additional throughput.





## Cached Reads

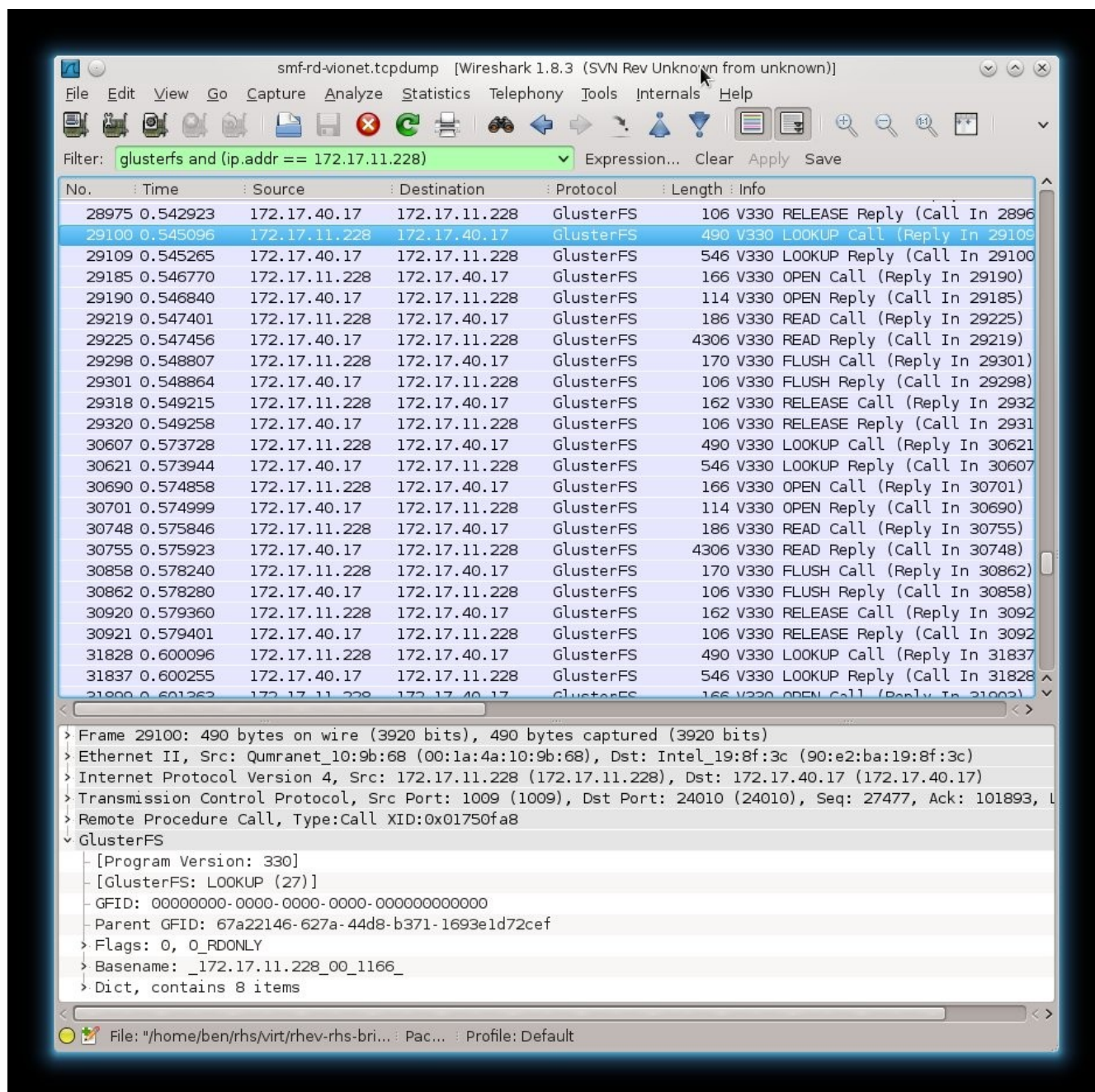


**Figure 4.4.2-7: Cached Read Results**

The cached read case has similarities to the uncached case. Peak throughput is achieved with eight hosts and eight servers, but throughput increases by more than 4x from the two host two server configuration. This result suggests that in the two VM two host configuration there were enough VMs to cause CPU resource starvation. After the VMs obtain sufficient CPU resources at four hosts (32 VMs/host), additional hosts do not help. Recall that each host has 24 cores. In the virtio-net case, there were  $32768 \text{ files/thread} \times 128 \text{ threads} = 4 \text{ million files} = 16 \text{ GB}$  of data total. This data should have been easily cacheable by a single Gluster server. The fact that virtio-net throughput with 16 hosts went from 7,000 to 20,000 files/sec as the servers scaled from from four to eight suggests that there was a limit on how many small file requests/sec Gluster could process. This limitation has been measured before and comparable numbers have been observed due to Gluster small file bottlenecks.



The next figure captures Gluster traffic for a single VM with a single thread reading 4-KB files.

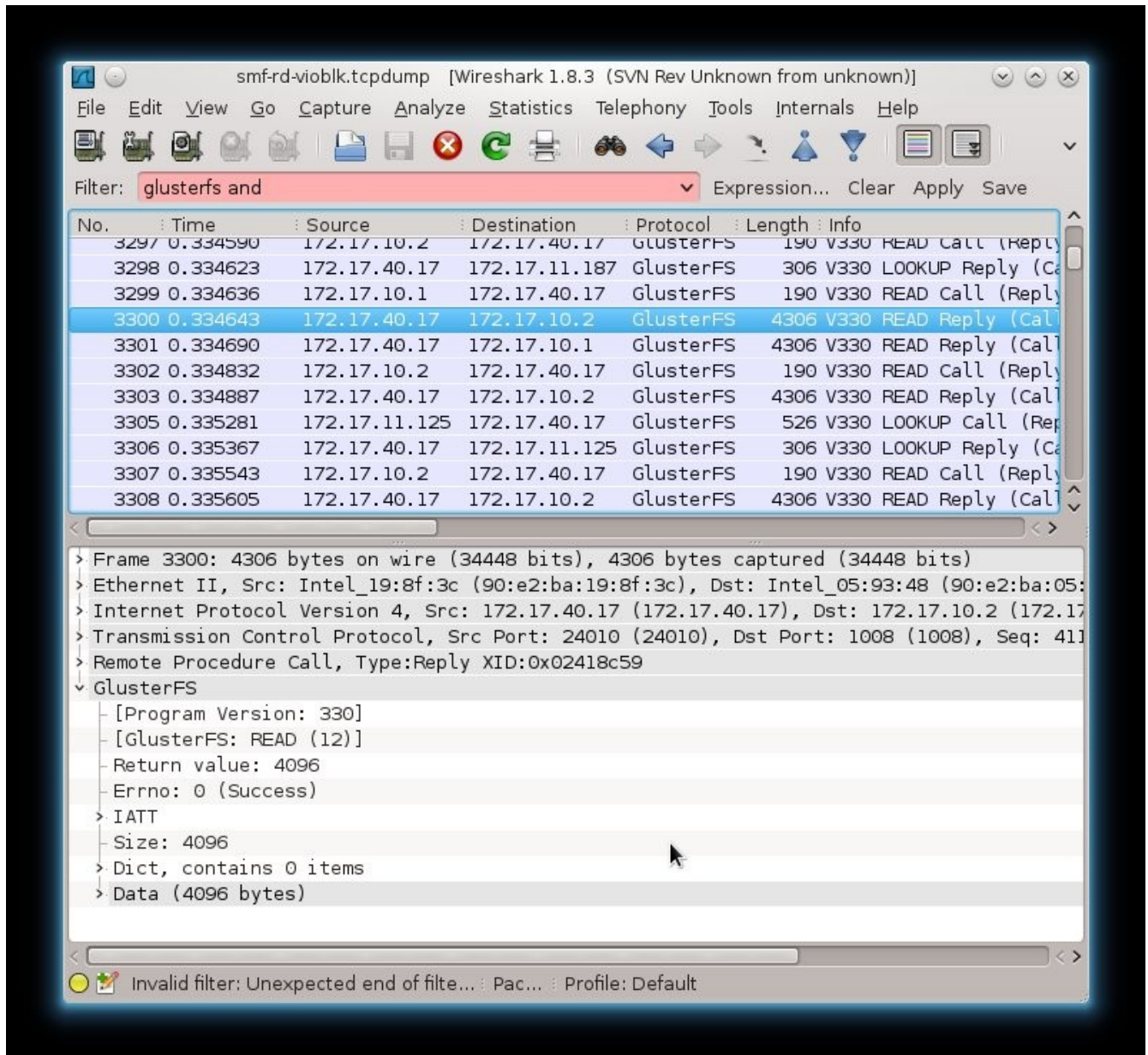


**Figure 4.4.2-8: Wireshark Data capture**

Each file requires metadata round-trips such as OPEN and RELEASE.



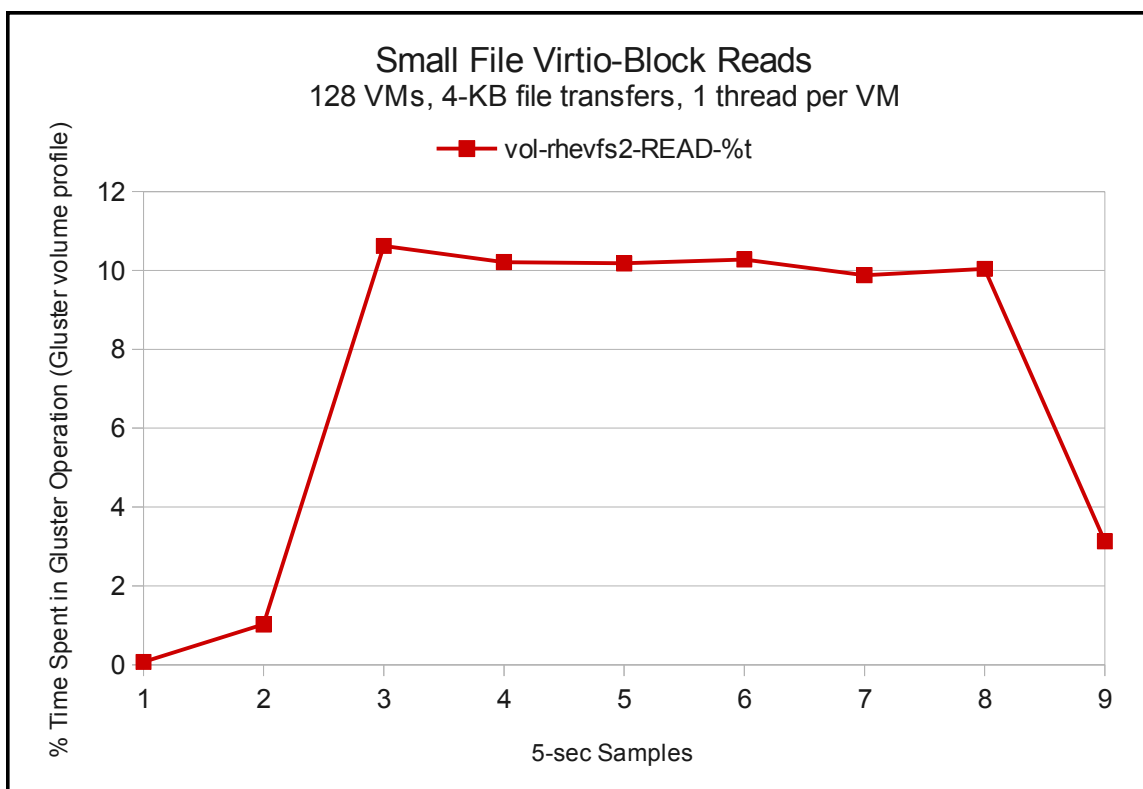
The next figure graphs the percentage of time was spent in Gluster operations for the virtio-net case. A very small percentage of time in the server is spent on reads and the majority of time is spent on file lookup.



**Figure 4.4.2-9: Wireshark Data Capture**



The following figure graphs how Gluster is reporting how the servers spend their time. There is only one significant operation occurring (LOOKUP was omitted as it was only 2%).



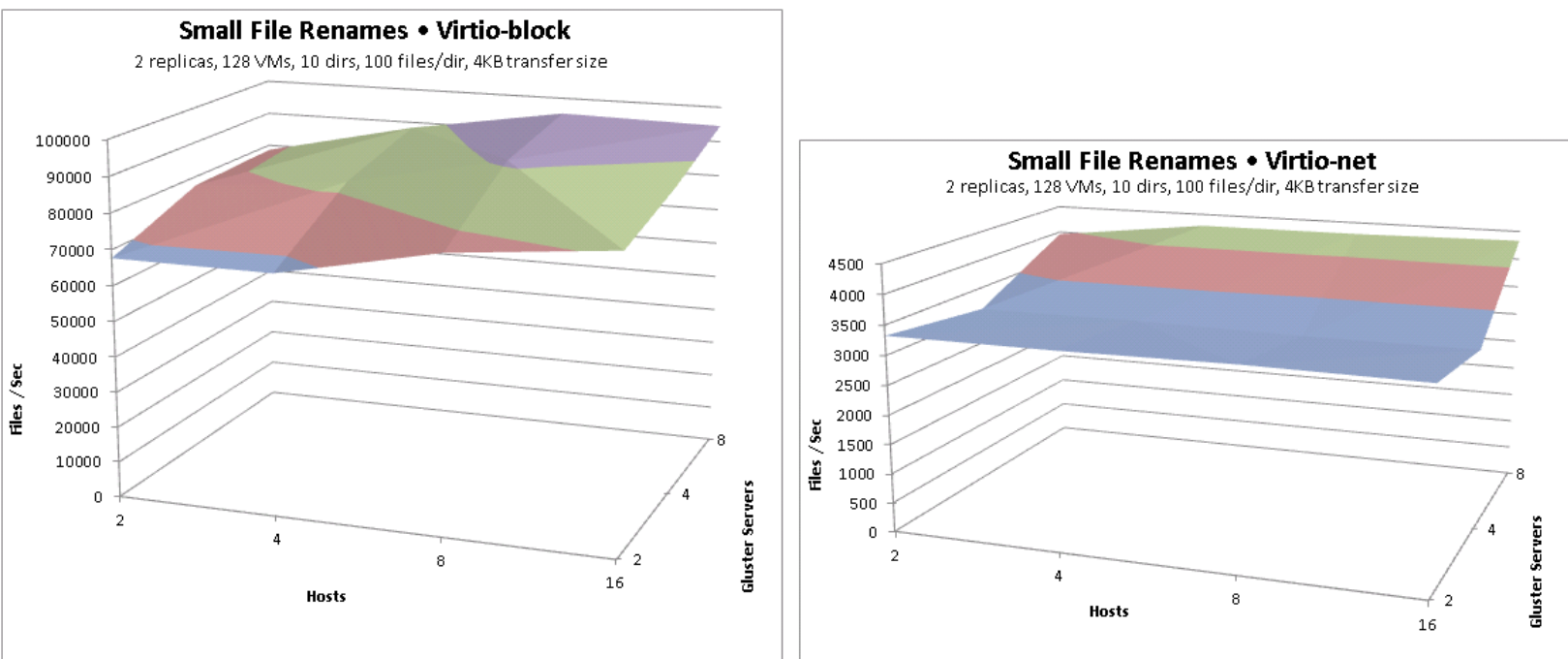
**Figure 4.4.2-10: Small File Virtio-Block Read Time**

In the above virtio-block graph, Gluster is only performing reads while all metadata is processed inside the VM, transparently to Gluster. This results in metadata caching and prefetching far better than what Gluster can presently do.





## Renames

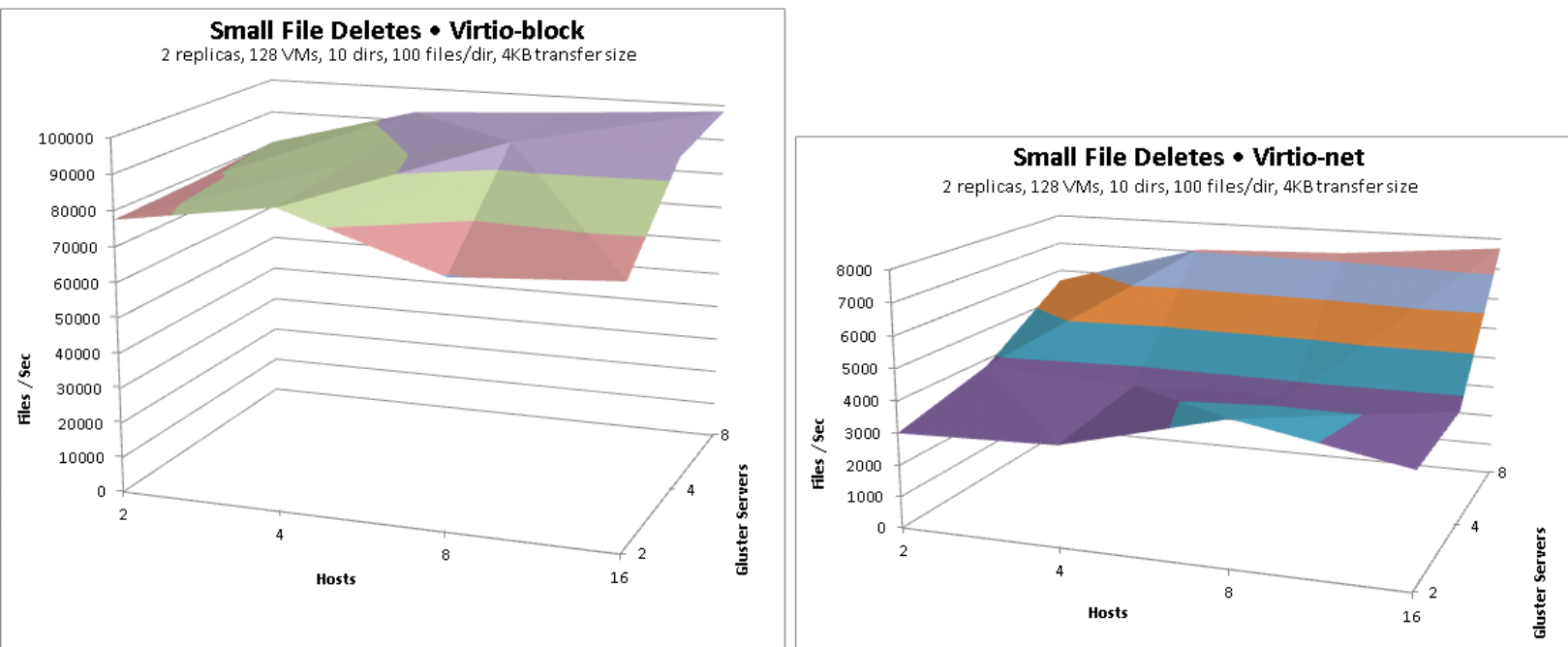


**Figure 4.4.2-11: Rename Results**

Virtio-block is roughly 20x faster than virtio-net in file renaming. For virtio-block or virtio-net, there is little scaling for the rename operation. For virtio-block, throughput actually decreases as the host count doubles from eight to 16. For virtio-net, as the number of Gluster servers increases from two to four, throughput decreases slightly (negative scaling) and does not linearly increase as servers increase from four to eight. This suggests that renames may be a problematic operation for large RHEV on RHS configurations.



## Deletes



**Figure 4.4.2-12: Delete Results**

For the above delete graphs, virtio-block has a 15:1 performance advantage over virtio-net although virtio-net starts to show some scaling increasing from four to eight servers, whereas virtio-block is not showing linear scaling as servers increase, and is showing actual decreases in throughput with low Gluster server counts. Note that these workloads are radically different in that virtio-block is doing random writes at the Gluster level, whereas with virtio-net Gluster is actually deleting individual files.



# Appendix A: RHEV GUI displays

## Data Center

Red Hat Enterprise Virtualization Manager Web Administration - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Red Hat Enterprise Virtualization... [+](#)

[https://gprfc025/webadmin/webadmin/WebAdmin.html#dataCenters-logical\\_networks](https://gprfc025/webadmin/webadmin/WebAdmin.html#dataCenters-logical_networks) [Google](#)

Most Visited [Zimbra Web Client ...](#) [Red Hat Bugzilla M...](#) [Red Hat](#) [linux](#) [netapp](#) [Gluster](#) [personal](#) [rhev](#)

**Red Hat Enterprise Virtualization** Logged in user: **admin@internal** | [Configure](#) | [Guide](#) | [About](#) | [Sign Out](#) **Feedback**

Search: DataCenter: [x](#) [★](#) [🔍](#)

**Data Centers** Clusters Hosts Storage Disks Virtual Machines Pools Templates Users Events

New Edit Remove Force Remove [Guide Me](#)

Name	Storage Type	Status	Compatibility Version	Description
gfs_D1	POSIX compliant FS	Non-Responsive	3.1	
gfs_D2	POSIX compliant FS	Up	3.1	

Storage Logical Networks Clusters Permissions Events

New Edit Remove

Name	Description
rhevm	Management Network
10GBrhevm	

Bookmarks

Tags

Last Message: [✓](#) 2013-May-19, 07:03:07 User admin@internal logged in. [Alerts \(6\)](#) [Events](#) [Tasks \(0\)](#)

Red Hat Enterprise Virtualization Manager Web Administration - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Red Hat Enterprise Virtualization... [+](#)

<https://gprfc025/webadmin/webadmin/WebAdmin.html#hosts-general> [Google](#)

Most Visited [Zimbra Web Client ...](#) [Red Hat Bugzilla M...](#) [Red Hat](#) [linux](#) [netapp](#) [Gluster](#) [personal](#) [rhev](#)

**Red Hat Enterprise Virtualization** Logged in user: **admin@internal** | [Configure](#) | [Guide](#) | [About](#) | [Sign Out](#) **Feedback**

Search: Host: [x](#) [★](#) [🔍](#)

**Data Centers** **Clusters** Hosts Storage Disks Virtual Machines Pools Templates Users Events

New Edit Remove Activate Maintenance [Configure Local Storage](#) [Power Management](#) [Assign Tags](#)

Name	Hostname/IP	Cluster	Data Center	Status	Load	Memory	CPU
gprfc004	172.17.10.4	gfs_C2	gfs_D2	Up	0 VMs	9%	
gprfc005	172.17.10.5	gfs_C2	gfs_D2	Maintenance	0 VMs	0%	
gprfc006	172.17.10.6	gfs_C2	gfs_D2	Maintenance	0 VMs	0%	
gprfc007	172.17.10.7	gfs_C2	gfs_D2	Maintenance	0 VMs	0%	

General Virtual Machines Network Interfaces Host Hooks Permissions Events

OS Version: RHEL - 6Server Active VMs: 0 Physical Memory: 48251 MB total  
Kernel Version: 2.6.32 - 279.2 Memory Page Sharing: Inactive Swap Size: 24191 MB total  
KVM Version: 0.12.1.2 - 2.2.5 Automatic Large Pages: Always Shared Memory: 0%  
VDSM Version: vdsmd-4.9.6-44.1 Number of CPUs: 12 Max free Memory for scheduling new VMs: 47930 MB  
SPICE Version: 0.10.1 - 10.el6 CPU Name: Intel Westmere CPU Type: Intel(R) Xeon Phi  
iSCSI Initiator Name: iqn.1994-05.c

**Action Items**  
[Power Management is not configured for this Host. Enable Power Management](#)

Bookmarks

Tags

Last Message: [✓](#) 2013-May-19, 07:03:07 User admin@internal logged in. [Alerts \(6\)](#) [Events](#) [Tasks \(0\)](#)



## Hosts

The screenshot shows the Red Hat Enterprise Virtualization Manager Web Administration interface in Mozilla Firefox. The browser address bar displays `https://gprfc025/webadmin/webadmin/WebAdmin.html#vms-general`. The interface is logged in as `admin@internal`. The **Hosts** tab is selected in the top navigation bar. The left sidebar shows a tree view with **System** expanded. The main content area displays a table of virtual machines:

Name	Host	IP Address	Cluster	Data Center	Memory	CPU	Network	Display	Status
beMyGuest098			gfs_C2	gfs_D2	0%	0%	0%		Down
beMyGuest099			gfs_C2	gfs_D2	0%	0%	0%		Down
beMyGuest100			gfs_C2	gfs_D2	0%	0%	0%		Down

Below the table, the **General** tab for `beMyGuest100` is selected, showing details:

- Name: beMyGuest100
- Description:
- Template: gluster5
- Operating System: Red Hat Enterprise
- Default Display Type: Spice
- Priority: Low
- Defined Memory: 512 MB
- Physical Memory Guaranteed: 512 MB
- Number of CPU Cores: 1 (1 Socket(s), 1 C
- Highly Available: No
- USB Policy: Disabled
- Origin: RHEV
- Run On: Any Host in Cluster
- Custom Properties: Not-Configured
- Cluster Compatibility Version: 3.1

The bottom status bar shows: Last Message: 2013-May-19, 07:03:07 User admin@internal logged in. Alerts (6) Events Tasks (0).

## Virtual Machines

The screenshot shows the Red Hat Enterprise Virtualization Manager Web Administration interface in Mozilla Firefox. The browser address bar displays `https://gprfc025/webadmin/webadmin/WebAdmin.html#vms-network_interfaces`. The interface is logged in as `admin@internal`. The **Virtual Machines** tab is selected in the top navigation bar. The left sidebar shows a tree view with **System** expanded. The main content area displays a table of network interfaces for the selected virtual machine:

Name	Network Name	Type	MAC	Speed (Mbps)	Rx (Mbps)	Tx (Mbps)	Drops (Pkts)	Port Mirro
10GBrhev	10GBrhev	Red Hat VirtIO	00:1a:4a:10:9b:51000	< 1	< 1	< 1	0	
rhev	rhev	Red Hat VirtIO	00:1a:4a:10:fe:ac1000	< 1	< 1	< 1	0	

The bottom status bar shows: Last Message: 2013-May-19, 07:03:07 User admin@internal logged in. Alerts (6) Events Tasks (0).





## Appendix B: Gluster Volume Information

Volume Name: rhevfs

Type: Distributed-Replicate

Volume ID: d762f1bc-bcd0-4f55-85ea-738d06f1131c

Status: Started

Number of Bricks: 4 x 2 = 8

Transport-type: tcp

Bricks:

Brick1: gprfs023-10ge:/mnt/brick0

Brick2: gprfs024-10ge:/mnt/brick0

Brick3: gprfs022-10ge:/mnt/brick0

Brick4: gprfs021-10ge:/mnt/brick0

Brick5: gprfs018-10ge:/mnt/brick0

Brick6: gprfs019-10ge:/mnt/brick0

Brick7: gprfs017-10ge:/mnt/brick0

Brick8: gprfs020-10ge:/mnt/brick0

Options Reconfigured:

performance.quick-read: off

performance.read-ahead: off

performance.io-cache: off

performance.stat-prefetch: off

cluster.eager-lock: enable

network.remote-dio: on

storage.linux-aio: off

performance.write-behind: on

diagnostics.latency-measurement: on

diagnostics.count-fop-hits: on

nfs.register-with-portmap: off

nfs.port: 6013

cluster.self-heal-daemon: on

nfs.disable: off



## Appendix C: timed-par-for-all.sh

```
#!/bin/bash
# execute a command in parallel on a set of host specified in param 1
# you must quote the command if it's more than a single word

logdir=/tmp/par-for-all.$$
echo "log directory is $logdir"
elapsed_times="$logdir/elapsed-times.list"
cmd="$2"
hostlist=$1
OK=0
SSH="ssh -o StrictHostKeyChecking=no -nTx "
#echo "starting in parallel on : "
rm -rf $logdir && mkdir -p $logdir
for n in `cat $hostlist` ; do
    #echo -n " $n"
    eval "time ( $SSH ${USER}@$n \"\$cmd\" > $logdir/$n.log 2>&1 ) 2>>$elapsed_times &"
    pids="$pids $!"
    pace.py 0.10
done
echo
j=0
host_array=( `cat $hostlist` )
for p in $pids ; do
    h=${host_array[$j]}
    wait $p
    s=$?
    chars=`wc -c < $logdir/$h.log`
    if [ $chars -ge 1 -o $s != $OK ] ; then
        echo
        echo "--- $h ---"
        if [ $s != $OK ] ; then
            echo "pid $p on host $h returns $s"
        fi
        cat $logdir/$h.log
    fi
    retcodes="$retcodes $s"
    (( j = $j + 1 ))
done
awk '/^real/ { print $2 }' $elapsed_times | tr 'sm' ' ' | awk '{print 60*$1 + $2}' | /shared/stats
for s in $retcodes ; do
    if [ $s != $OK ] ; then exit $s ; fi
done
exit $OK
```